3-21-2019

Comparison of Novel Heuristic and Integer Programming Schedulers for the USAF Space Surveillance Network

Kanit Dararutana

Comparison of Novel Heuristic and Integer Programming Schedulers for the USAF Space Surveillance Network

THESIS

Kanit Dararutana, Capt, USAF

AFIT-ENS-MS-19-M-108

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-MS-19-M-108

COMPARISON OF NOVEL HEURISTIC AND INTEGER PROGRAMMING SCHEDULERS FOR THE USAF SPACE SURVEILLANCE NETWORK

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Kanit Dararutana, B.A., B.S., M.B.A.

Capt, USAF

21 March 2019

AFIT-ENS-MS-19-M-108

COMPARISON OF NOVEL HEURISTIC AND INTEGER PROGRAMMING SCHEDULERS FOR THE USAF SPACE SURVEILLANCE NETWORK

THESIS

Kanit Dararutana, B.A., B.S., M.B.A.
Capt, USAF

Committee Membership:

Dr. Bruce A. Cox, Ph.D.
Chair

Dr. Raymond R. Hill, Ph.D.
Reader

Mr. David Meyer, M.S.
Reader

Abstract

Space is a highly congested and contested domain begetting the importance of prioritizing the Space Situational Awareness (SSA) mission. With increased dependence on space assets, scheduling and tasking of the Space Surveillance Network (SSN) is vitally important to maintaining space dominance. According to the 2004 USSTRATCOM Strategic Directive 505-1 (SD 505-1) the SSN uses centralized tasking, with decentralized scheduling. Enhancing SSA within available resources is paramount, and the development of a centralized SSN scheduler to maximize performance is crucial. This research develops and compares novel scheduling models to a model reecting the 2004 SD 505-1. Novel schedulers were developed to reduce time gaps between observations, prioritize high value space objects, and retain maximum observation quality. In both single and multi-sensor scenarios, these novel schedulers maintained the same, or higher, levels of observation threshold retention in high priority targets, while increasing observation threshold gains in lower categories. Simulations using the novel schedulers showed dramatic improvement, especially in multi-sensor scenarios, in the mean and maximum time between observations of sample space objects compared to the SSN Scheduler Model. Novel schedulers showed at least a 3% improvement in meeting threshold requirements, a 12% decrease in mean time between observations, and up to a 9% decrease in maximum time between observations. Finally, these benefits were realized with a nominal increase in processing time for most novel schedulers. Results of this research can educate national policy makers on the benets of proposed upgrades to current and future SSA systems.

To my Mother, Father, and Sisters

Acknowledgements

I greatly appreciate the guidance and mentorship I received from my faculty advisor, Lt Col Bruce Cox. I would also like to thank my committee members, the faculty and student members of the Center for Space Research and Assurance, and the space modeling and simulation team, without whose support the work accomplished in this thesis would not have been possible. I am grateful to the staff at Analytical Graphics Incorporated for the educational use of their STK Engine. Most importantly, I would like to thank my loving wife for supporting and encouraging me as I worked through this endeavor.

Kanit Dararutana

Table of Contents

Page

Page

Page

List of Figures

List of Tables

List of Abbreviations

A3C	Asynchronous Advantage Actor-Critic
ACS	Ant Colony System
AFRL	Air Force Research Laboratories
AFSSS	Air Force Space Surveillance System, or, Space Fence
AGI	Analytical Graphics Incorporated
ALTAIR	ARPA Long-Range Tracking and Identification Radar
ASAT	Anti-Satellite
BMEWS	Ballistic Missile Early Warning System
COEs	Classical Orbital Elements
CSpOC	Combined Space Operations Center
DA	Direct Ascent
DQL	Deep Q Reinforced Learning and Distributed Q Learning
DSRC	Defense Supercomputing Resource Center
Elset	Element Set Classification
GEO	Geostationary Orbit
GEODSS	Ground-Based Electro-Optical Deep Surveillance
GPO	Geosynchronous Polar Orbit
GSO	GEO Orbit over the Equator
GUI	Graphical User Interface

HEO	High Earth Orbit/Highly Elliptical Orbit
HPC	High-Performance Computer
JFCCSpace	Joint Functional Component Command for Space
JSpOC	Joint Space Operations Center
LEO	Low Earth Orbit
MEO	Mid Earth Orbit
MIT	Massachusetts Institute of Technology
MOSS	Moron Optical Space Surveillance
MOTIF	Maui Optical Tracking and Identification Facility
MSSS	Maui Space Surveillance System
PARCS	Perimeter Acquisition Radar Attack Characterization System
PG	Policy Gradient
RSO	Resident Space Objects
RST	Report Style Template
SBSS	Space-Based Space Surveillance
SNR	Signal-to-Noise Ratio
SSA	Space Situational Awareness
SSN	Space Surveillance Network
SST	Space Surveillance Telescope
STKE	Systems Tool Kit Engine

STK	Systems Tool Kit
TCP/IP	Transmission Control Protocol/Internet Protocol
TLE	Two-Line Element Set
USSTRATCOM	United States Strategic Command

COMPARISON OF NOVEL HEURISTIC AND INTEGER PROGRAMMING SCHEDULERS FOR THE USAF SPACE SURVEILLANCE NETWORK

I. Introduction

1.1 Background

One of the missions the United States Air Force provides under Air Force Space Command is Space Situational Awareness (SSA) in order to monitor and track the ever-growing collection of objects orbiting Earth. After more than 50 years of space development, congestion between objects in space has become a major concern and continues to become worse as more nations become increasingly dependent on space assets and launch new systems into space [7]. As of 2015, there were 1,000 active satellites, 7,000 inactive satellites, and numerous pieces of space debris. Many pieces of space debris are too small to track and it is estimated that more than 20,000 pieces of 1 cm to 10 cm and approximately 200,000 pieces smaller than 1 cm exist [8]. Keeping up with the exponentially increasing number of space objects, improving capabilities to detect objects currently too small to track, and providing appropriate intelligence on objects of concern is vitally important to the DoD. However, increasing the number of DoD assets may not be feasible in the future with a shrinking budget. Instead, current assets, processes, and software must be examined to determine where advances may be made at lower monetary costs.

1.2 Problem Statement

The United States' Space Surveillance Network (SSN) is currently tracking and maintaining orbital information on objects in all of the various orbits. With limited resources and aging technology, the current assets within the SSN's inventory, comprised of earth-based optical telescopes, space-based optical telescopes, and radar tracking sites, cannot keep up with an increasingly difficult mission. The issue is evident with situations such as the Chinese anti-satellite (ASAT) test on Fengyun-1C in January of 2007 which generated nearly 2700 pieces of debris greater than 10cm, or the collision between Iridium 33 and Cosmos 2251 in February of 2009, which generated over 1600 pieces of debris greater than 10cm. The Fengyun-1C event is also estimated to have produced over 150,000 pieces of debris larger than 1cm, all of which are capable of disabling spacecraft [3]. If the SSN cannot completely perform its mission, the US is left vulnerable due to gaps in intelligence resulting from an inability to properly characterize resident space objects (RSO)[8]. Due to shrinking budgets and limited resources, the DoD needs to find ways to optimize the usage of current assets without adding to the inventory.

1.3 Research Questions

- Can a scheduling method be developed which outperforms the scheduler represented in 2004 SD 505-1 at reducing mean and maximum time between observation for a population of RSO's while meeting RSO observation thresholds?
- Can a novel scheduler be developed that improves upon the scheduler used in prior research with roughly same processing requirements?

1.4 Approach

This research develops and compares novel scheduling techniques to a scheduler model reflecting the 2004 USSTRATCOM Strategic Directive 505 Volume 1 (SD 505-1) [6], the most recent redacted edition of the document that provides direction on SSA scheduling processes and tracking. By implementing various scheduling techniques, the research determined if changes to the scheduling process results in improvements to areas such as lowering the mean age of data across RSO categories, and increasing the number of observed satellites in lower priority categories without reducing prioritized asset uncertainty. The research began by leveraging prior research models, utilizing a purely greedy heuristic prioritizing time between observations, to create a model scheduler mirroring that described in SD 505-1. Prior thesis [9] [10] [11] research in this area was limited to GEO, the models developed in this research expand this scope to include RSO's in LEO, MEO, and HEO. This baseline model was compared against four different novel schedulers, two built off modifying the previously developed heuristic scheduler, one using binary integer programming, and the final one a binary integer program using multiple objectives. Following testing, comparative analysis was performed to determine how these schedulers performed.

1.5 Summary

Chapter II details current SSA background as well as orbital fundamentals necessary to addressing asset concerns, and the current SSN doctrine. It also gives background on scheduling theory and relevant scheduling techniques as well as briefly discuses previous efforts to optimize SSA. Chapter III introduces novel heuristic schedulers expanding on research presented in Stern and Wachtel, Felten, and Batemen's theses [9] [10] [11]. Chapter III also introduces novel binary integer optimization

based schedulers. Chapter IV details analytical results of these schedulers, to include the baseline scheduler and sensitivity analysis. Chapter V provides conclusions and recommendations for future work.

II. Literature Review

2.1 Overview

This chapter begins by examining recent Air Force doctrine and detailing the importance of Space Situational Awareness (SSA). It continues by outlining the basics of orbital motion and trajectory, along with effects that cause orbits to change, as well as how to read the Two-Line Element Set. The chapter then introduces signal-to-noise ratio and how it is calculated, followed by a description of RSO tracking methodology. Next, the chapter details information about the Space Surveillance Network (SSN), sensors within its inventory, and how tasking is performed through the SSN. This is followed by a review of scheduling theory as well as several scheduling techniques to include Zero-One Linear Programming, Multi-Objective Optimization, greedy heuristic, Genetic Algorithm, and Reinforcement Learning. Finally, the chapter concludes by examining previous solution efforts to optimize SSA scheduling, to include several related thesis papers.

2.2 Space Situational Awareness

Joint Publication (JP) 3-14, Space Operations, defines SSA as, the requisite foundational, current, and predictive knowledge and characterization of space objects and the [Operating Environment] upon which space operations depend - including physical, virtual, information and human dimensions - as well as all factors, activities, and events of all entities conducting, or preparing to conduct, space operations [2]. SSA allows for "the enabling of a description of the location and operation of US space assets as well as the location and function of the assets of other nations, particularly those that are, or could become, our enemies" [12] as well as characterizing the threat created by the growing uncatalogued space debris population [3]. In other words,

SSA is crucial in securing the safety of space assets as well as protecting military and national interests through the collection of all available information about the current space environment.

Because the US and many other nations heavily rely on space assets, "SSA is a necessity for any nation that seriously bases its military and economic well-being even partly on space capabilities" [12]. World nations are developing two separate types of space related capabilities. The first involves deploying military and economic space assets to include communications, positioning, and detection missions. The second includes technologies designed to counter or otherwise deny other nations' assets. Both categories can be threatening to US National Security and establish the need to provide accurate location and orbital information through SSA [12]. The USAF divides SSA requirements into five pillars that include 1) Detect, Track and Identify, 2) Characterization, 3) Tactical Warning and Attack Assessment, 4) Data Integration and Exploitation, and 5) Spacecraft Protection and Resiliency [8]. This research focuses on detecting, and subsequently tracking, resident space objects (RSO's).

2.2.1 Common Orbits

Space object orbits are classified depending on their distance from the Earth's surface and the shape of their orbit. Common orbit types include Low Earth Orbit (LEO), Medium Earth Orbit (MEO), Highly Elliptical Orbit (HEO), and Geostationary Orbit (GEO), shown in Figure 1.

GEO objects orbit the Earth at the same rate at which the Earth rotates upon its axis, tracing a figure eight over the ground, i.e. ground trace, or in the case of a special type of GEO orbit over the equator-called geostationary (GSO), which appear with a ground trace of a fixed point. GEO orbits allow for consistent line of sight observations available from one-third of the Earth and are ideal for worldwide

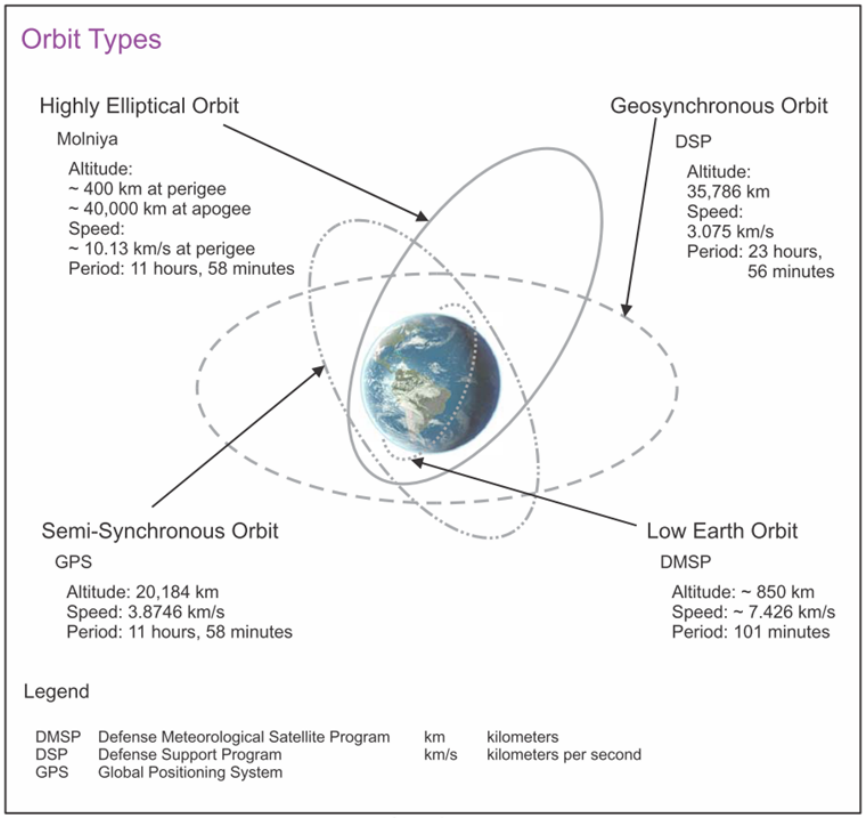

Figure 1. Orbit Types (not to scale) [1]

communications, surveillance, reconnaissance, observing large-scale weather patterns, and missile warning [2].

HEO objects orbit in the shape of a long ellipse with their most distant points from the earth (called apogee) up to 25,000 miles away and their closest points (called perigee) as close as a few hundred miles. The nature of their orbits cause a long dwell time near apogee, making these orbits ideal for communications, scientific, surveillance, and weather missions over higher latitudes [2].

MEO orbits include orbits between that of LEO and GEO, and have no formal altitude, but a special case exists with a semi-synchronous MEO orbit when they are near circular. This orbit allows an object to repeat an identical ground trace after two, 12 hour, revolutions and is most well-known for the orbit utilized by the Global Positioning System (GPS) as well other communication missions [2].

LEO orbits are objects relatively close to the Earth with an average orbit time of approximately 90-100 minutes, allowing for less powerful transmitters and the ability to achieve higher resolution imagery, making LEO ideal for Intelligence, Surveillance, and Reconnaissance (ISR) missions, environmental monitoring, scientific, manned space-flight, and small communications satellites [2]. The short overhead view period necessitates a constellation of many satellites evenly spaced in several orbital planes in order to maintain continuous coverage.

A brief synopsis of the characteristics for each orbit type is in Figure 2.

Orbit Type and Characteristics				
Type	Description	Advantages	Disadvantages	Applications
Geosynchronous Earth orbit	Roughly circular ~23,000 miles above Earth's surface	Continuous coverage over specific area Coverage nearly hemispheric	Far from Earth - resolution and signal limitations Easier to jam signal latency	Communication Surveillance Reconnaissance Weather collection Missile warning
Highly elliptical orbit	Long ellipse ~600 miles at perigee (closest to Earth) ~25,000 miles at apogee (farthest from Earth)	Long dwell time over a large area Coverage of high North or South latitudes	Continuous coverage requires multiple satellites	Communication over high North or South latitudes Scientific Surveillance Reconnaissance Missile warning
Medium Earth orbit	Roughly circular Between ~1,000-22,000 miles above Earth's surface	Stable orbit Less signal latency	Highest radiation level environment	Positioning, navigation, and timing Communication
Low Earth orbit	Roughly circular Up to ~1,000 miles above Earth's surface	Near Earth - high resolution and signal strength	Small coverage area over Earth surface Limited coverage windows for any specific geographic region	Surveillance Reconnaissance Weather collection Manned space flight Communications

Figure 2. Orbit Type and Characteristics [2]

2.2.2 Orbit Changes

Typically, an RSO's orbit remains constant unless the RSO causes the movement, through maneuvers commonly called Delta-V's, or the object is affected by an exter-

nal force, a perturbation, or by interacting with another object, such as a collision. Maneuvering in space costs fuel and decreases the life of a satellite. Perturbations can cause an object, which otherwise would not change, to alter its orbit and can include effects such as the gravitational pull of the Earth, Sun, Moon, and other planets. The variance in Earth's gravitational field can cause perturbation effects. Additionally, perturbations can be caused by atmospheric drag, the Sun's solar pressure, and interactions between solar radiation and the Earth's magnetic field [2].

Numbers of both operational spacecraft and orbital debris are increasing greatly each year as more spacecraft are launched into orbit, orbital debris collisions continues proliferating additional debris, and previously undocumented debris from previous collisions becomes catalogued [3] as shown in Figure 3.

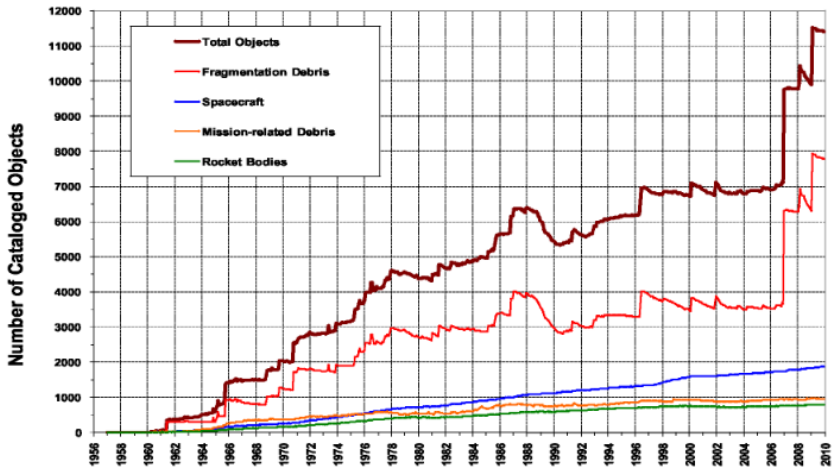

Figure 3. Growth of the Catalogued Satellite Population. Note that some debris from the major breakups of 2007 and 2009 have yet to be officially cataloged. [3]

As Lt. Gen. Helms says, "Operating within the increasingly congested, contested, and competitive space environment requires strategically reexamining our processes, planning flexibility, awareness of the space environment, and collaboration efforts with all space faring nations and corporations" [2]. These factors increase the importance of quality SSA, an accurate accounting of where objects are, and satellite conjunction

analysis, understanding the risk of collision between a satellite and another object in orbit [13].

2.2.3 Two-Line Element Set

In order to describe a satellite's orbit around the Earth, NORAD developed and NASA adopted the Two-Line Element Set (TLE) Format, which allows outside applications to compute positioning at any moment of a tracked satellite. The TLE contains descriptive information for an individual satellite as well as Keplerian orbital elements of a satellite. The first line contains information such as Satellite Catalog number, Element Set (Elset) Classification, International Designation, Elset Epoch in UTC or the time in which the TLE was taken. The second line contains the six Classical Orbital Elements (COE's) including the semi-major axis-defining the size of the orbit, the eccentricity-giving the shape of the orbit, the inclination-representing the orientation of the orbit with respect to the Earth's equator, the argument of periapsis-defining the perigee of the orbit with respect to the Earth's surface, the longitude of the ascending node-representing the location of the ascending and descending orbit locations with respect to the Earth's equatorial plane, and the true anomaly at epoch denoting where the satellite is within the orbit with respect to perigee [4]. An example of a TLE is shown in Figure 4.

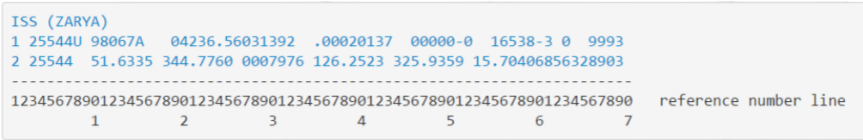

Figure 4. International Space Station TLE [4]

2.2.4 Detection

The ability to distinguish an object from its background is the definition of detection regarding RSO's. To do so, the signal-to-noise ratio (SNR) must be at least 2.5 in order to successfully detect the object[14]. However, users may specify higher SNR values for successful detection, as Koblick, Goldsmith, Klug et al. [15] and Stern and Wachtel [9] used a more conservative SNR of 6. This research utilized an SNR threshold of 4 to account for errors made in sensor performance assumptions like Basraoui [16], who continued Stern and Wachtel's research, and researchers at Georgia Institute of Technology, who studied optimization of CubeSat detection [17]. This thesis utilizes the following equations as described in Stern and Wachtels [9] to calculate SNR. Equation 1 details the amount of reflected light received from a Lambertian Sphere using phase angle and the Lambertian reflection coefficient [18]

$$\psi = \frac{2C_d}{3\pi^2}(sin(\alpha) + (\pi - \alpha)cos(\alpha)) \tag{1}$$

where

ψ is the fraction of the maximum received light

α is the phase angle

C_d is the Lambertian reflection coefficient.

Equation 2 indicates the power emitted by a RSO based on its size and solar radiation [19].

$$P_{em} = S\pi r_{RSO}^2 \tag{2}$$

where

P_{em} is the power emitted

S is the solar radiation

r_{RSO} is the radius of the RSO.

Equation 3 indicates the power received by the sensors detector based on power emitted, reflected light, and the area of the observing sensor [19].

$$P_{RCVD} = \frac{\psi P_{em} \tau_{opt} \tau_{atm} A_{RCVR}}{R^2} \tag{3}$$

where

P_{RCVD} is the Power received by the detector

R is the distance from RSO to the observer

τ_{opt} is the atmospheric transmittance

τ_{atm} is the optics transmittance

A_{RCVR} is the sensor collecting area.

Using the power received, an output signal is determined in Equation 4 [19].

$$Signal = \frac{P_{RCVD} \lambda_{avg} \eta}{hc} t_{int} \tag{4}$$

where

$Signal$ is output signal

λ_{avg} is the average of the wavelengths of the bandpass

η is the efficiency of the detector turning received signal into output

t_{int} is the integration time

h is Plancks constant

c is the speed of light.

Noise is the unpredictable fluctuation in signal output as a result of uncertainty in electron generation in the detectors electronics [9] and can be modeled using Kriscinunas and Schaefers [20] model generated by the sky and moon in Equations 5-10.

$$I^* = 10^{-0.4(3.84+0.026|a|+4\times10^{-9}a^4)} \tag{5}$$

$$f(\rho) = 10^{5.36}[1.06 + cos^2(\rho)] + 10^{6.15-\frac{\rho}{40}} \tag{6}$$

$$X(Z) = (1 - 0.96sin^2Z)^{-0.5} \tag{7}$$

$$B_{moon} = f(\rho)I^*10^{-0.4kX(Z_m)}[1 - 10^{-0.4kX(Z)}] \tag{8}$$

$$Rad_{sky} = B_{moon} + B_{zen} \tag{9}$$

$$N_{sky} = \frac{Rad_{sky}A_{RCVR}\tau_{opt}\eta\lambda_{avg}t_{int}IFOV^2}{hc} \tag{10}$$

where

I^* is the illuminance of the moon outside of the atmosphere

a is the lunar phase

ρ is the angle between sky/target position

$f(\rho)$ is the Rayleigh (molecule) scattering

Z is the zenith angle

$X(Z)$ is the Moon distance

B_{moon} is the brightness of the moon

B_{zen} is the moonless zenith sky brightness

k is the atmospheric extinction coefficient

Z_m is the lunar zenith angle

Rad_{sky} is the total sky radiation

N_{sky} is the sky noise

$IFOV$ is the sensors Instantaneous Field of View.

Finally, Equation 11 for SNR is reached using Howells [21] model:

$$SNR = \frac{Signal}{\sqrt{Signal + \eta(N_{sky} + N_d)t_{int} + N_r^2}} \tag{11}$$

where

N_d is the dark signal property

N_r is the sensor read noise property.

Using the above calculations, an SNR above a user defined value indicates that a sensor has the ability to feasibly make an observation of that RSO [9].

2.2.5 Tracking

Once an object has been detected, a TLE can be developed and the RSO can be tracked. Tracking involves determining and maintaining RSO orbital parameters in order to determine current and predict future locations. Ideally, RSO tracking should be accomplished frequently enough such that the interval of observations is short enough at least to allow future observations without the efforts of initial detection. Tracking of RSO's assists in learning the objects maneuverability as well as potentially the object's mission [22]. At least three observations, each creating two data points, are required in order to use Gauss' Method to solve for the six Classical Orbital Elements (COEs) used to characterize the orbit of an RSO [23]. Using Gauss least squares method, a best fit approximation is calculated based on multiple possible orbits created from the observations due to uncertainty [22]. Horwood et al determined that high-confidence orbit estimates can be made based on observations spaced between 0.2 and 0.6 orbital periods if at least three or four observations are made in total, where more observations within shorter periods beget smaller confidence estimates [24]. As time from last observation progresses, orbit confidence decreases and the need to perform additional observations become more important. To maintain high-confidence orbit estimates, the Combined Space Operations Center

(CSpOC) specifies an ideal number of observations for each object based on orbit and priority [25].

2.3 Space Surveillance Network

Joint Functional Component Command for Space (JFCC Space) utilizes the Space Surveillance Network (SSN) in order to detect, track, and characterize RSO's and maintain intelligence on high-interest targets. The SSN is a collection of 47 optical and radar sensor systems spread across 31 geographic locations on Earth as well as two orbital regimes, all of which are divided into three categories: dedicated, collateral, and contributing [26]. Radar sensors within the SSN are either mechanical, employing a mechanical antenna tracker which sends a beam of radar energy to the target is reflected and returned to the radar receiver for measurement, or phased-array which steer radar energy electronically rather than mechanically move an antenna. Mechanical radars are limited by their ability to only track one object at a time and cannot efficiently search for targets due to their single radar beam. Phased-array radars can track numerous targets simultaneously but are limited by power available, the high cost of construction and complex maintenance. Optical sensors, more specifically electro-optical, form images by gathering light waves reflected off an object into electrical impulses recorded onto magnetic tape. Optical sensors are limited by their reliance on light, cannot track during the day or under overcast sky conditions, and the object must be reflecting light [27].

Dedicated sensors are those operationally controlled by US Strategic Command (USSTRATCOM) whose primary mission is to provide SSA and include Ground-Based Electro-Optical Deep Space Surveillance (GEODSS) systems, Moron Optical Space Surveillance (MOSS), Eglin AFB AN/FPS-85 phased-array radar [27], Space Surveillance Telescope (SST), Space-Based Space Surveillance (SBSS) [9], and the

Air Force Space Surveillance System (AFSSS), or Space Fence, which was shut down in 2013 [28]. Collateral sensors are those controlled by USSTRATCOM whose primary mission is not space surveillance but characterize RSO's as a result of providing for their primary mission and include the Ballistic Missile Early Warning System (BMEWS), Maui Optical Tracking and Identification Facility (MOTIF), MAUI Space Surveillance System (MSSS), PAVE PAWS, Perimeter Acquisition Radar Attack Characterization System (PARCS), Antigua Radar, Ascension Radar, and Kaena Point Radar [27]. Finally, contributing sensors include those owned and operated by agencies other than USSTRATCOM that provide SSA capabilities as a secondary mission by request from the Combined Space Operations Center (CSpOC) previously known as the Joint Space Operations Center (JSpOC) [29]. These sensors include Millstone/Haystack owned and operated by Lincoln Laboratories of the Massachusetts Institute of Technology (MIT), ARPA Long-Range Tracking and Identification Radar (ALTAIR) operated by the U.S. Army, and Cobra Dane operated by Raytheon [27]. This research focuses on a theoretical sensor whose capabilities mimic GEODSS with a 1 m aperture and a 2.15 m focal length providing a two-degree field of view which operates only at night [8].

Data collected from each of these sensors are directed to the CSpOC to determine TLE's for tracked objects maintained in the satellite catalog [27]. In order to maintain the satellite catalog accuracy, the CSpOC determines the number of tracks necessary for each satellites requirements and allocates SSN resources to track those satellites. Each satellite is described by a numeric category defining its relative priority (categories 1 to 5 from highest to lowest) and with an alphabetic suffix defining the number of tracks per day to collect based on suffix (A through U) [25]. The 2004 Strategic Command Directive 505-1 Vol 2 [6] outlines these suffixes and their associated metric taskings, reflected in Table 1.

Table 1. Metric Tasking Suffixes [6]

Suffix	Mechanical Radar	Phased Array Radar	Optical
A	All Possible $\leq$ 50 per pass	All Possible $\leq$ 50 per pass	All Possible $\leq$ 50 per pass
B	10 per pass	10 per pass	10 per pass
C	5 per pass	5 per pass	5 per pass
D	3 per pass	3 per pass	3 per pass
E	1 per pass	1 per pass	1 per pass
F	6 per pass 2 passes per day	3 per pass 2 passes per day	8 per pass 2 passes per day
G	6 per pass 1 pass per day	3 per pass 1 pass per day	8 per pass 1 pass per day
H	6 per pass 1 pass per day	3 per pass 1 pass per day	8 per pass 1 pass per day
J	3 per pass 2 passes per day	1 per pass 2 passes per day	5 per pass 2 passes per day
K	3 per pass 1 pass per day	1 per pass 1 pass per day	5 per pass 1 pass per day
L	3 per pass 1 pass per day	1 per pass 1 pass per day	5 per pass 1 pass per day
M	6 per pass 1 pass per day	3 per pass 1 pass per day	8 per pass 1 pass per day
N	6 per pass 1 pass per day	3 per pass 1 pass per day	8 per pass 1 pass per day
O	9 per pass 2 passes per day	9 per pass 2 passes per day	9 per pass 2 passes per day
P	3 per pass 1 pass per day	1 per pass 1 pass per day	5 per pass 1 pass per day
Q	3 per pass 1 pass per day	1 per pass 1 pass per day	5 per pass 1 pass per day
R	10 per pass All passes	10 per pass All passes	16 per pass All passes
S	10 per pass All passes	10 per pass All passes	16 per pass All passes
T	10 per pass All passes	10 per pass All passes	16 per pass All passes
U	5 per pass 2 passes per day	5 per pass 2 passes per day	5 per pass 2 passes per day

The CSpoC generates a centralized tasking plan and then individually tasks SSN sensors. Each sensor then individually schedules how it will attempt to track its assignments. Currently, centralized SSN scheduling is infeasible due to communication and technology limitations [25]. Each sensor attempts to meet its tracking allocations

and the results are sent back to CSpOC for evaluation of the overall sensors response to the taskings, the catalog is updated, and the cycle is repeated [25].

2.4 Scheduling Theory

Each individual sensor develops a schedule to meet the tracking assignments it has received. These schedules are driven by the importance of the satellite, the object priority which is influenced by the age of previous TLE data, and the track requirements based on object suffix [25]. Developing an ideal schedule falls under an important branch of operations research, called Scheduling Theory, which studies situations of "optimal distribution and sequencing of the jobs of a finite set to be processed on either a deterministic single machine or in a multi-machine system under different assumptions on the nature of this processing" [30], where a job may be observing an RSO and a machine may be a resource such as a ground sensor. Scheduling is a decision-making process regularly used to determine an optimal allocation of resources to jobs within a given time period in order to optimize one or many objectives [31] and is typically concerned with finding the appropriate sequence of tasks on resources such that both all constraints are met and some performance criterion is optimized [32]. A schedule consists of each job to be performed on each resource at each appropriate time in order [33].

Often, scheduling of jobs is subject to a set of constraints which take the form of three primary types: technological, precedence, and resource. Technological constraints require processing jobs through resources in specific order [32]. For example, to manufacture a ruler, first wood must pass through a resource, the cutting machine, where it is cut to size, after that it goes through a resource to have markings printed on it, and finally pass through a resource to have the wood treated and sealed. Because satellite observations are considered single jobs performed by each sensor, technolog-

ical constraints do not apply to a sensor-satellite scheduling problem as satellites do not need scheduling to different sensors in specific orders [33]. However, precedence constraints require processing certain jobs in specific order [32]. Using the previous example, a precedence constraint may be that 6 inch rulers must run through the marking resource before 12 inch rulers, both having passed through the cutting resource already. In a sensor-satellite scheduling problem, precedence constraints occur as a result of satellite priorities, where a satellite of the highest priority must be seen before a satellite with the next highest priority [33]. Resource constraints exist when one or more resources are limited [32], in the case of the ruler manufacturing, X rulers may require construction, but only one cutting resource exists. In the case of sensor-satellite scheduling, resource constraints exist when only a select number of sensors are visible to a satellite at a given time. Additionally, another resource constraint exists with the number of satellites a sensor may observe at the same time [33].

Assuming one or more feasible schedules exist, the main objective for a scheduler is to determine which of those schedules is optimal based on defined measures of effectiveness (MOE). For example, many schedulers attempt to minimize the total time required to perform all jobs, or to accomplish the most jobs in a given period of time [32]. The sensor-satellite schedule, in general, falls into the latter example where ideally the number of observations should be maximized in a 24 hour period [33]. However, other schedules may be developed to fulfill different objectives such as minimizing the mean age of a collection of satellites or maximizing the observations of a specific category of satellites.

Schedulers may solve scheduling problems using a number of different methods and solution techniques. For many complex scheduling problems, a single universally efficient solution technique does not exist. Instead, the selection of the appropriate solution technique depends on the nature of the scheduling model [33]. The develop-

ment of a schedule for a sensor, or set of sensors, to observe a fixed group of satellites may be developed using different heuristic methods or scheduling algorithms. Each method may provide different results for various objectives and decision makers must decide which approach best suits their needs. The following section contains the description of several techniques used to solve scheduling problems, as well as summaries of studies using those approaches to develop optimal solutions for the RSO-to-sensor-scheduling problem.

2.4.1 Binary Linear Programming

Linear Programming is a class of mathematical problems concerned with minimizing or maximizing a linear function subject to a set of linear equality or inequality constraints [34]. In 1947, while working as a mathematical advisor to the USAF comptroller, George B. Dantzig conceived the general class of linear programming problems. A Soviet mathematician and economist, L.V. Kantorovich formulated and solved a problem of this type in 1939, however, his work did not become known until 1959. Linear programming got its name due to the USAF referring to plans and schedules as "programs" leading Dantzig to publish his first paper on the subject as "Programming in a Linear Structure," which was later coined "linear programming" in 1948 by economist and mathematician T. C. Koopmans [34]. The general form for a linear program is:

$$\text{maximize} \qquad c_1 x_1 + ... + c_n x_n \tag{12}$$

$$\text{subject to} \quad a_{11} x_1 + ... + a_{1n} x_n \leq b_1, \tag{13}$$

$$\vdots$$

$$a_{m1} x_1 + ... + a_{mn} x_n \leq b_m, \tag{14}$$

$$x_n \geq 0, \forall n \in N \tag{15}$$

where there are n variables and m constraints. In a linear program, the constraints are linear functions of the variables. If desired, minimization may be accomplished by reversing the signs of $c_1 ... c_n$. In 1949, Dantzig published the "simplex method" for solving linear programs, which still enjoys wide acceptance, though other methods for solving linear programs now exist [34]. A type of linear program where all variables are integers is called Integer Linear Programming, or Integer Programming, and further restriction of variables to that of 0 or 1 is Zero-One, or Binary Linear Programming. The sensor-satellite scheduling problem can be modeled as a Binary Linear Program, as the choice for a particular satellite observed by a particular sensor at a point in time is reflected by 0-not observed, or 1, for observed [34].

2.4.2 Genetic Algorithm

An increasingly popular meta-heuristic optimization method is the Genetic Algorithm, first developed by Holland in 1960 as part of his artificial intelligence research [35]. Genetic Algorithms are based on Darwin's theory of evolution, where each solution returns a fitness score and more 'fit' solutions within a population are more likely to survive to the next iteration and be combined, thereby passing on their 'genes.' If a child inherits the more desirable traits from each parent, it should be more fit than the parents. Over time and generations, the population as a whole should move toward higher levels of fitness [35].

The Genetic Algorithm starts by mapping the problem solution space into a genetic string and randomly creating an initial population of feasible solutions. Each string, or solution, is evaluated using a fitness function, and the best strings 'reproduce' and produce offspring for the next generation. Each string reproduces in proportion to its fitness value such that higher ranked strings have a greater chance of

reproducing than lower ranked strings. During "mating," two 'parent' strings impart a portion of their strings to the child, in a process called crossover, allowing for the creation of new child strings [35].

The fitness evaluation and mating is repeated, those strings with above-average fitness tend to survive, while those below-average die out. In order to maintain diverse populations, mutations are introduced randomly. A typical Genetic Algorithm repeats the three steps (evaluation/selection, crossover, and mutation) for each generation until terminated by some condition, like a certain number of generations. Typically, Genetic Algorithms converge on a fitness score [36], which may be the global optimal solution, or local optimal solution. To control convergence, experimenters may adjust population size, crossover type and probability, mutation type and probability, selection operators, and number of generations [35].

Using this method, Andreas Hinze and Hauke Fiedler et al [37] compared two Genetic Algorithms, one single-objective and the other multi-objective, to provide RSO catalog maintenance. In simulations containing 762 MEO and GEO objects using the German Space Operation Center telescope. The single-objective Genetic Algorithm focused on reducing position error covariance, or the uncertainty of a given RSO's tracklet, using a method called Shannon Information Content [37]. The Genetic Algorithm then provided a fitness value with

$$\text{maximize } F = \max \sum_{j=1}^{N} SIC_j. \tag{16}$$

The multi-objective algorithm included detection probability provided by the pre-estimated size magnitude of the RSO as the second objective [37]. Here, smaller

magnitudes, m, are more ideal and therefore the second objective function was

$$\text{minimize } F = \min \sum_{j=1}^{N} m_j. \tag{17}$$

The results of both algorithms converged after approximately 500 generations to a solution scheduling 182 observations, with the multi-objective Genetic Algorithm slightly outperforming the single-objective [37]. Their research demonstrated the applicability of Genetic Algorithms for creating schedules optimizing both on single and dual objectives. However, their research did not take into consideration the fact that RSO's are not homogeneous but have differing priorities. In the Air Force operating environment, certain RSO's are always of crucial intelligence value and therefore require a higher priority for observation and tracking versus other objects. Thus, including prioritization among objectives within scheduling is as, or more important than reducing uncertainty and detection probability.

2.4.3 Reinforcement Learning

In 1959, Arthur Samuel coined the term "Machine Learning" in his journal article, *Some Studies in Machine Learning Using the Game of Checkers*, which covered the first computer learning program playing checkers, improving each time it played [38]. Ethem Alpaydin defines machine learning as "programming computers to optimize a performance criterion using example data or past experience" [39]. Machine learning may be applied in a number of ways. In learning associations, a program determines connections between two entities, say consumers and products, to determine what patterns occur, like a consumer who buys ketchup often also buys mustard. Another Machine Learning application is classification, where a program may predict what categories an entity falls under, for instance given many pictures, it may be able to distinguish between a cat and a dog. Similarly, regression uses training data, where a

program may be able to predict a response value fitting a function to the data to find the response. These three tasks are considered supervised learning because learning is conducted by mapping inputs to outputs with an external operator providing correct outputs. Unlike the previous three, unsupervised learning only receives input and the goal is to find regularities or clusters in the input [39]. Reinforcement learning is a final application, where there is also no external source providing correct information. In order to obtain substantial reward, the program must prefer solutions it has found effective in the past and must discover new solutions on its own through discovery by trying solutions it has not tried before [40].

Regan [41] capitalized on machine learning techniques to develop a modular neural network for tasking and scheduling of SSA systems. This modular neural network method involved developing three separate neural networks to perform searching for new objects, reacquisition of known objects, and tasking the overall system to search or reacquire at a specific time. Each neural network branch was trained individually through deep Q reinforcement learning (DQL), a form of reinforcement learning which builds a matrix called a Q-table and uses values from that table to make policy choices which optimize actions based on values learned through state-action relations [41]. Then, the networks are integrated to perform a basic SSA-like mission. This system was tested using a single observer, representing a sensor, to track 50 objects in a 10x10 unit region with random initial positions and velocities. The objects traveled linearly across the environment and the overall network commands the observer to search from one location to another in order to minimize age of the data collected and position error of the targets [41]. The results showed after 100 iterations of training the tasking network, there was a 15% increase in performance and after 250 iterations a 25% increase in performance, demonstrating the network's ability to learn and optimize in a short period. Regan's research is limited in scope as it is

designed to model a basic SSA-like mission with limited objects and a single sensor and provides a simple interpretation of the SSA problem [41]. Further development utilizing this model with comparison to scheduling architectures in place evaluating the same criteria with larger sets of objects and sensors would provide great insight into performance and potential.

A different deep reinforcement learning method by Linares and Furfaro [42] is implemented through Asynchronous Advantage Actor-Critic (A3C), a policy gradient (PG) method which leverages the advantage of being able to perform in parallel and therefore on a network of resources. Here, the reinforcement learning technique is demonstrated on a system of 100 and 300 RSO's of varying types (LEO, MEO, GEO, etc.) with a single ground sensor trained in 10,000 steps. The results show the solutions to both simulations allow object covariance to converge from an initial 100km to below 10 km in 4 hours, and 10 hours, respectively. Demonstrating the system's ability to learn and reduce covariance error [42]. Similar to Hinze and Fiedler, this approach includes a single objective to reduce covariance, but does not indicate whether high priority RSO's are maintained or considered. Like Regan, this method would benefit from comparison to current architectures in a multi-sensor, large RSO-count simulation, considering multiple objective, that decision makers require, would also prove beneficial.

Such a comparison was completed in a study by Bryan Little and Carolin Frueh [43] comparing the distributed Q learning (DQL) to Ant Colony System (ACS) optimization meta-heuristic and a greedy algorithm, ACS is shown to outperform both the greedy and DQL solutions. In a system of 512 GEO objects using a single ground sensor, the ACS solution observes 508 objects, followed by 496 and 425 for greedy and DQL respectively when given a cost function with the single objective to maximize weighted viewing directions. Of note with this study is the computation time

required for each solution, where the DQL solution took nearly 8 times as long as the greedy, and the ACS 22 times as long as the greedy. The DQL system is expected to perform better in the future as it had not been fully trained properly [43]. In a second simulation including uncertainty as a multi-objective problem, the ACS solution outperforms greedy 445 objects to 384 [43]. This comparison is completed only on GEO objects and a single ground sensor, while the 2004 SD 505-1 indicates sensors within the SSN schedule separately, by allowing central completion of the schedule across multiple sensors, it is possible to achieve a solution with better performance results.

2.4.4 Multi-Objective Optimization

In some cases, planning and scheduling problems involve more than one objective. For example, in sensor-satellite scheduling not only is maximizing the number of observations desired, but also minimizing the mean time since the TLE was last updated for the collection of satellites. Often, these objectives conflict with one another, producing different solutions for each separate objective [5]. One basic method to optimize multiple objectives simultaneously involves the Weighted Sum Method where the following problem is solved:

$$\text{minimize} \qquad \sum_{i=1}^{k} w_i f_i(\mathbf{x}) \tag{18}$$

$$\text{subject to} \quad \mathbf{x} \in S \tag{19}$$

where there are i objectives and weights $\sum_{i=1}^{k} w_i = 1$. In these problems, a unique solution may or may not exist but a set of mathematically equivalent solutions may be determined [5]. Here, the solutions may look like Figure 5, where Z represents the set of possible solutions in objective function space, the dark black line represents the

set of solutions called the Pareto Frontier, or Pareto Optimal Set, and $z*$ represents a Pareto optimal objective vector [5].

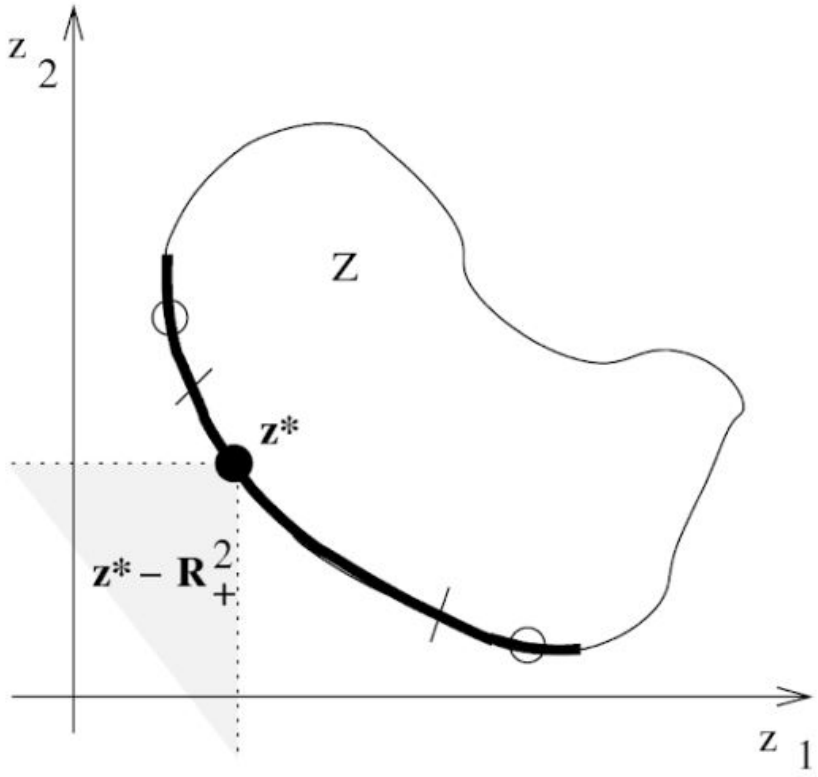

Figure 5. Set of Pareto optimal solutions [5]

The Pareto Optimal Set is named for Vilfredo Pareto, who determined that **x** is Pareto Optimal if variables are allocated so that changing the allocation would not decrease some criterion without at the same time increasing at least one other criterion [44]. Typically, solving multi-objective problems involve a decision maker, who specifies a preference in weights w_i, and may be presented with this set of solutions for their decision. The Weighted Sum Method provides a simple technique to find a set of Pareto optimal solutions, however, a disadvantage is that it only truly works in a convex solution space such as Figure 5. In a non-convex space it may only find a portion of the Pareto optimal solutions [5]. However, for the sensor-satellite problem, finding a portion of the Pareto optimal solutions is satisfactory as they are all

mathematically equivalent solutions.

2.4.5 Greedy Heuristic

Some problems are so complex that optimal solutions are difficult to find, and may take an enormous amount of effort and time to complete. One method to solve such problems is to use heuristics to produce good, if not optimal, solutions. Heuristics are simple yet useful tools to solve problems and aid decision making or discovery. "Heuristic" comes from the Ancient Greek verb heuriskein, meaning "to find out" or "to discover" [45]. They are prescriptive procedures which detail how an acceptable solution may be found given constraints like limited time. In doing so, heuristics may become efficient tools that may not provide the optimal solution but may come close and do so expeditiously. Examples of heuristics to solve problems date back far as the thirteenth century, where Catalan philosopher Raimundus Lullus developed a mechanical device able to generate all debate argument combinations of religious or philosophical attributes automatically [45]. In modern optimization, heuristics can be traced back to 1945, where mathematician George Plya's procedures, such as decomposing a problem to work toward a solution and later recombining the problem, would later contribute to fields like Artificial Intelligence [45].

One well known heuristic is the greedy heuristic. The greedy heuristic always makes the locally optimal choice. The hope is that this choice and subsequent locally optimal choices eventually lead to a globally optimal solution, however greedy heuristics do not always yield optimal solutions [46].

Three related thesis papers by Jordan Stern and Steven Wachtel [9], Michael Felten [10], and Mark Bateman [11] combined both multi-objective optimization and Genetic Algorithms to solve a different aspect of SSA-to identify optimal combinations of ground and space-based sensors required to perform the GEO SSA mission.

Using Air Force Research Laboratory (AFRL) Defense Supercomputing Resource Center's (DSRC) High-Performance Computer (HPC), combinations of ground and space-based sensors, simulated by Systems Tool Kit (STK) Engine were run in parallel to simultaneously produce multiple performance metrics on each architecture, providing necessary information to determine sensor to RSO visibility over given periods of time. In post processing, a schedule was created using a greedy algorithm to observe 813 GEO RSOs over a 24-hour period [9]. The greedy heuristic first identified lists of possible observation intervals for available RSO's with cloud free line of sight for each sensor. Then, for each sensor the RSO visible to that sensor with the oldest satellite data was determined, assigned for observation, and its' data age reset to zero. The entire process was repeated for every 30 second time interval in a 24-hour day. The end result was a schedule which tasked each sensor during each time interval, provided an RSO was visible. After the schedule was created, performance metrics were taken for detection based on mean RSO diameter, tracking through use of the mean wait time between observations, and a total cost figure derived for the SSA architecture set up. These three performance metrics were combined into a fitness score for each architecture and used by a Genetic Algorithm to identify near-optimal architectures [9].

Felten's research expanded upon Stern and Wachtel's by including access to a 12-satellite Geosynchronous Polar Orbit (GPO) constellation, twilight imaging on current ground sensors within the architecture, increasing space-based sensors in a plane from 4 to 6, and alternative optimization techniques to include Simulated Annealing and Particle Swarm Optimization [10]. In Batemans research, the architecture was expanded to incorporate monitoring of direct ascent (DA) vehicles and Model Based Systems Engineering practices in optimization [11]. Their research proved parallel simulation methodology with a multi-objective Genetic Algorithm could be used to

converge on architecture solutions [10]. Each of these research efforts utilized the same greedy scheduler which selected the RSO with the largest time since observation and assumed adequate SNR was achieved. Additionally, they focused on only GEO RSO's and modeled a system where scheduling could be completed centrally, with each sensor working in unison rather than separately. The optimal architectures derived from the assumptions in these models would not reflect operating conditions under the 2004 SD 505-1.

2.4.6 Merit-based Greedy Scheduler

Another Air Force Institute of Technology related thesis by Walid Basraoui [16] reflected on Stern and Wachtels work by implementing a merit-based scheduler. The scheduler collected data as in the previous thesis, but differed from prior thesis by assigning each RSO one of three priority levels. These priorities, combined with the RSOs SNR value, and time between observations, were used to determine a Figure of Merit for each sensor-RSO pair during each time interval. The weights between the three values depended on the weight allocated to the priority and the remainder split between SNR and time between observations. Similar to the previous greedy scheduler, for each interval the scheduler stepped through each sensor within the architecture to find the RSO with the highest figure of merit, assigned that sensor for observation during that time interval, and reset the time between observations. This was repeated for every time interval producing a schedule. This scheduler was tested with the same 813 GEO satellites but with a multi-domain SSA architecture to include 10 ground telescopes, 4 equatorial Low Earth Orbit (LEO) satellites and 3 GEO satellites. The scheduler was capable of reducing mean maximum observation time gaps for satellites of the highest importance from 81 min to 53.7 min, but the result increased mid-range satellites from 81 min to 160.8 min and the lowest priority

satellites from 81 min to 868.3 min [16]. This scheduler combines multiple objectives with the greedy heuristic, but does not model the RSO priority conditions described in 2004 SD 505-1, nor does it take into consideration observation threshold requirements, allowing the opportunity for revisiting high priority RSO's more frequently than necessary at expense of lower priority RSO's.

2.5 Conclusion

Improving the SSN is a critical task for the DoD as is ensuring SSA provides the required intelligence to act in the ever-growing and contested space environment. Without that vital data and without necessary development, US space assets become vulnerable. The most comparable approaches to this research were the theses by Stern and Wachtel [9], Felten [10], Bateman [11], and Basraoui [16]; however, each study limited RSO's to GEO objects, utilized a greedy scheduler focused only on reducing the largest time between observations, and assumed a centralized scheduler. This research expands RSO's to all orbit classes, demonstrates various schedulers, against baseline operations as detailed in 2004 SD 505-1 and includes focus on priority, uncertainty, and SNR. Finally, this thesis compares centralized scheduling to decentralized.

Modern day SSA utilizes a centralized prioritization list tasking completion objectives to individual sensors, while delegating actual scheduling to those individual sensors. Studies by Hinze and Fiedler [37], Regan [41], and Little and Frueh [43] provide promising optimization techniques, but do not include prioritization as an objective. Linares and Furfaro [42] simulate a cohesive tasking mechanism, but they do not go beyond single objective optimization. This thesis explores the impact of multi-objective optimization schedulers.

III. Methodology

3.1 Overview

This chapter discusses the overarching problem formulation, model development, and explanations for each scheduler implemented. The chapter explains the methods used to find measurement values for all resulting schedules. Areas of measurement include mean and max age between observations, total number and sum of RSO's observed by each priority category, and total number of RSO's which met target threshold values in one day. The chapter concludes with an explanation of the experimental trials conducted.

3.2 Materials and Equipment

The following section details materials and equipment used to carry out this research:

3.2.1 Python 2.7.1

Python 2.7.1 is a free open source, commonly used programming language. Using Spyder as the editor, a script can be written and fed via connect to STK. This allows a user to code loops to create multiple objects efficiently. By utilizing Python scripts and connect, scenarios are run much faster and data can be generated in mass amounts. Python was also used to model scheduler algorithms for comparison in this research.

3.2.2 PuLP

PuLP is a free open source Linear Program modeler written in python. PuLP was used in python code with scheduler algorithms that included linear program models.

3.2.3 Systems Tool Kit (STK)

Analytical Graphics Incorporated (AGI) Systems Tool Kit (STK) 11.4 was utilized to model the scenario and generate the access data for the ground site. STK is a physics based software tool which models the Earth, that allows engineers and scientists to perform complex analyses of ground, sea, air, and space assets and can produce various reports such as phase angles, zenith angles, azimuth and elevation angles, and range for analysis. AGIs connect commands were scripted and fed via python using a Transmission Control Protocol/Internet Protocol (TCP/IP) socket.

3.2.4 Previous Thesis

Data generation and original scheduler code were retrieved and modified from Bateman's thesis [11], which originated from Stern and Wachtel's research [9], for initial comparison. Equations and some assumptions built into their code were carried over into initial models.

3.3 Approach Overview

To compare 2004 SD 505-1 scheduler against novel schedulers a common environment was developed and utilized. STK was used to model the environment, schedule codes matching Bateman's [11] greedy scheduler and the SSN scheduling process according to 2004 SD 505-1 were created and compared against two novel schedulers. The first novel scheduler, the Relaxed SSN Scheduler Model, relaxed constraints within the SSN model, while the second, the Relaxed SSN Scheduler Model with Spacing, added spacing between observation intervals to maximize beneficial geometries. Two schedulers using binary integer programming were also created and compared to the above heuristic schedulers.

3.4 Data Generation

3.4.1 Python and STK Connection

Architectures were evaluated via STK 11.4 simulation. In addition to the more traditional STK Graphical User Interface (GUI), STK can be controlled through a module called Connect, which receives commands from Python (or other languages) in the form of strings through a TCP/IP connection. A python generation script is used to control STK which received the architecture parameters and number of RSO's as inputs. The script created sub-directories for the Report Style Template (RST) files and output reports, then created the Report Style Template (RST) files, which describe the contents and format of a report and were necessary for moon phase, lunar zenith angle for the ground telescope, lunar phase angle and target zenith angle for every RSO/ground telescope pair, and solar phase angle for every RSO/sensor platform pair because STK does not generate them by default. RST files were saved and uploaded later to the script, where STK was commanded to create a scenario in the time period selected.

3.4.2 RSO Generation

The RSO population was generated using Two-Line Elements (TLE's) for the entire available catalog, downloaded from Joint Functional Component Command for Space at www.space-track.org, translated into usable format using Microsoft Excel, and saved in Python as a list which was imported into the data generation script. Data was downloaded on 2 Oct 2018 and was comprised of 3,968 RSO's. For each scheduling model mentioned later, additional data fields were created to accompany the set of RSO's. The first field retained randomly generated priority categories, according to distribution given by AFSPC. Only suffixes A-E were modeled in this research,

per recommendation by Mr. Francis G. Lundy, 18th Space Control Squadron, on November 2nd, 2018, who also provided the following estimations of percentages for RSO categories featured in Table 2.

Table 2. RSO Priority Category Percentages

Category	% of Space Catalog
1	0.01 %
2	20 %
3	5 %
4	25 %
5	50 %

The second field contained the calculated SNR ratio for each RSO to the ground sensor. The third field contained randomly generated interval values to represent starting time between observations. These values were generated between zero and the average highest values observed of the sample selection accessed during the catalog download. 3,968 RSO's were available to model for this research. The final field was a counter for each RSO starting at zero detailing the number of times the RSO observed.

3.4.3 Ground Sensor Generation

For this research, the ground sensor characteristics mimicked that of GEODSS sensor at Socorro, NM (Latitude 33.82, Longitude -106.66, Altitude 1403m) [26]. The ground-based telescope was created within STK with solar exclusion angle of 40, lunar exclusion angle of 10, minimum elevation angle of 20 and constrained to only operate in umbra, as in previous thesis [9]. To further mimic the GEODSS sensor, the simulation allowed the sensor to view 3 RSO's at any given point in time to simulate the three telescopes available at that location. Next, custom vectors and angles were created for every RSO/sensor platform pair to include view vector, solar phase angle between view vector and RSO-sun vector, RSO perspective view vector, Lunar Phase Angle, the angle between the View Vector and the Lunar phase

Angle, the angle between the View Vector and the default RSO/Moon vector, and the Lunar Zenith Angle. The information was saved and a "Moon Phase" report, access reports, and Azimuth-Elevation-Range (AER) reports were created for every RSO/sensor platform pair. The desired single access duration for this model was set to 30 seconds, assuming this would be enough time to allow for slewing and settling the sensor between observations, and capturing multiple images.

3.4.4 Model Simulation Periods

For the scenario, three different 24 hour periods were selected for comparison. Both the 24hr periods reflecting summer solstice and vernal equinox were chosen to represent worst case and average case scenarios, as in previous thesis[9], while winter solstice was added as a best case scenario comparison, therefore the simulation dates were 21-21 Jun 2019, 20-21 Mar-2019, and 21-22 Dec 2019 from 0000Z to 0000Z respectively. Since the access duration was set to 30 seconds, each 24 hour duration was divided into 2,880 30 second intervals.

3.5 Base Model Greedy Scheduler

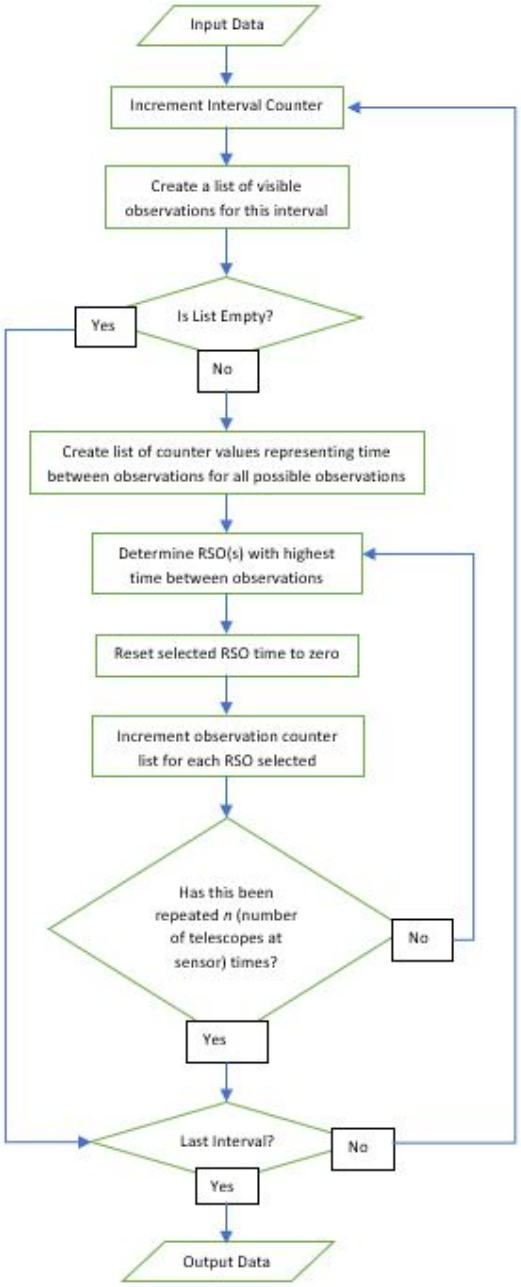

Figure 6. Base Model Greedy Scheduler Flow Chart

In order to see how prior research schedulers performed against novel schedulers developed for this research, a baseline model was created. Figure 6 is the flow chart

representing Stern and Wachtel's [9] original greedy scheduler as coded by Bateman [11]. In essence, this scheduler first determined what observations were available to the sensor at the given point in time and, of those available, determined the RSO with the highest time between observations for selection. The scheduler repeated this process for every interval.

The scheduler uses the access reports from STK to determine a list of observation intervals for each sensor/RSO pairing for which an observation was possible. An additional list n_{RSO} long is created and initialized to zero to function as a counter through the time period. The scheduler then loops during each time period and searches through RSO's visible to the sensor and selects the RSO with the highest counter value, or the first within the list in the case of ties. The counter element for the corresponding RSO is set to 0, simulating a successful observation. The sensor-RSO pair element within the schedule list is set to 1, while all others for that sensor during that time period are set to 0. The end result was a full 24-hour schedule for the sensor that tasked the sensor if there was a visible RSO. Although Stern and Wachtel use SNR ratio for their fitness evaluations of their Genetic Algorithm, the SNR ratio is not calculated for the scheduler and any RSO in range is assumed to have an acceptable SNR value of 6 [9].

3.6 SSN Scheduler Model

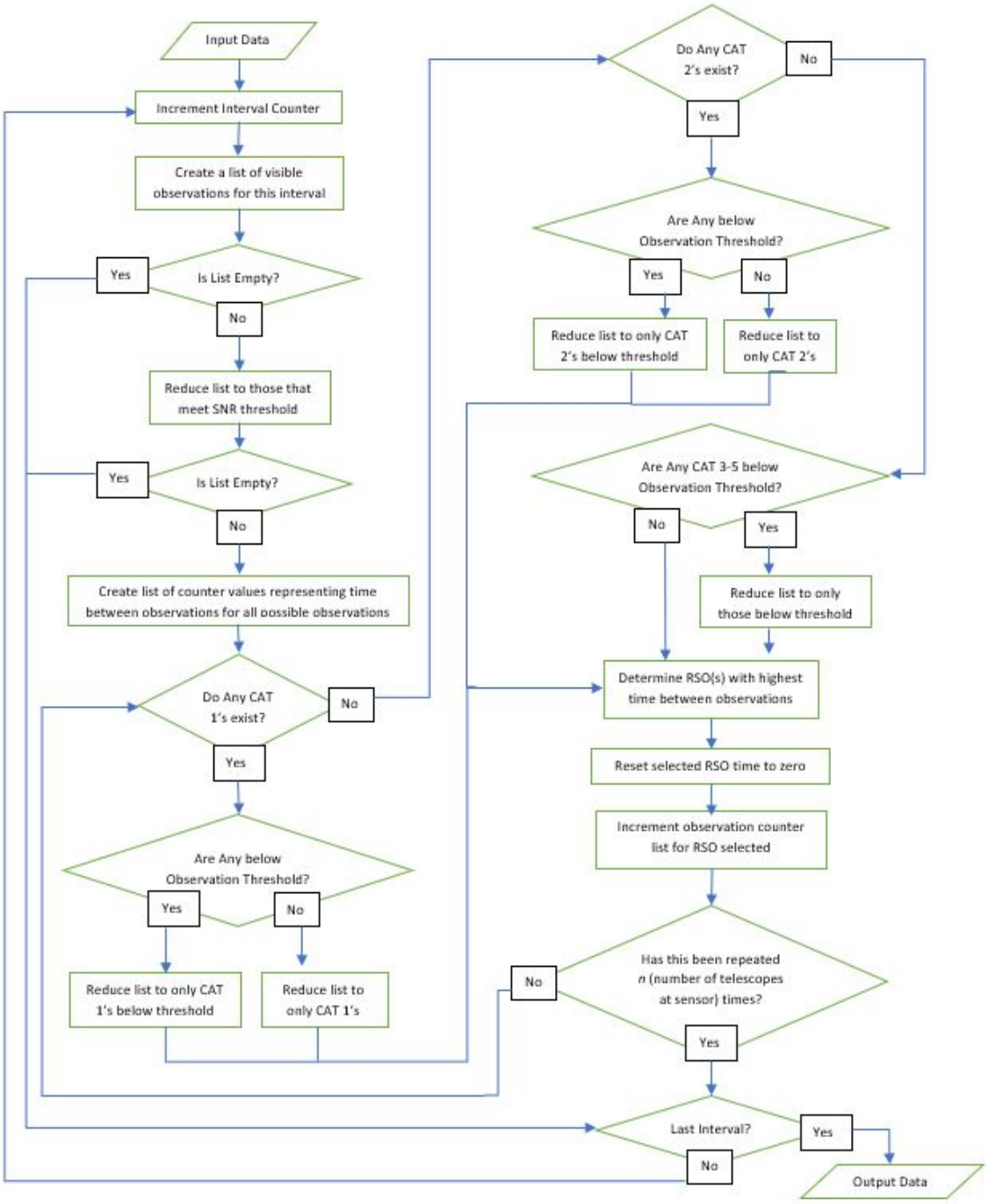

Figure 7. SSN Scheduler Model Flow Chart

The SSN Scheduler Model represents an RSO scheduler as set by the 2004 SD 505-1. The intention of SSN Scheduler Model is that if a Category 1 is observable, it must be observed. If no Category 1's are visible, then if a Category 2 is observable, it must be observed. Otherwise, the scheduler proceeds to fulfill category 3-5 thresholds and reduce the RSO's that have not been observed in the longest. The flow chart is seen in Figure 7.

Of all RSO's in range with SNR ratio's which meet thresholds, if there are Category 1 RSO's present below their desired observation threshold, then the sensor must observe them first, starting with the one with the highest time between observations. The SNR ratio check ensures the sensor is only attempting to observe RSO's for which it has a good chance to successfully receiving track data. If all Category 1's have met observation thresholds, then the Category 1 with the highest time between observations is selected. In this way, Category 1's are observed before any other category RSO and those below observation threshold are fulfilled first. If no Category 1's are in range with an SNR ratio above threshold, the process is repeated for Category 2 RSO's. If neither Category 1's nor 2's are in range with SNR's that meet threshold, the scheduler repeats the process for Category 3-5's again first looking for RSO's that have not met observation thresholds yet, selecting the highest time between intervals, and otherwise selecting the highest time between intervals available of any RSO observable.

After generating the list with possible observations, the scheduler checks whether the SNR value for that RSO meets requirements and reduces the observable list. Then, the scheduler checks through the list of accessible RSO's for the sensor at this interval and searches for the highest priority, starting at category 1 Suffix A. If the scheduler finds at least one, it cuts down the list of possible observations to only of the highest category available. If the highest category available is any category 1 or

2, the scheduler progresses to the next step. If the highest category is 3-5, it creates a list of those accessible RSO's with category 3-5 and create that list indicating whether or not the RSO has yet made threshold values. Then, it reduces the list of considered RSO's for observation down to only those that have not yet met threshold. Finally, no matter the category, the scheduler searches for the RSO with the highest time between observations and assign the sensor to observe only that RSO at that interval and resetting that RSO's counter value and updating the list counting how many times it has been observed that day. If all category 3-5 RSO's met threshold for the day, it chooses the RSO with the highest time between observations. The scheduler then loops through each time interval, incriminating all counters, and repeat the same process for the new time interval.

3.7 Relaxed SSN Scheduler Model

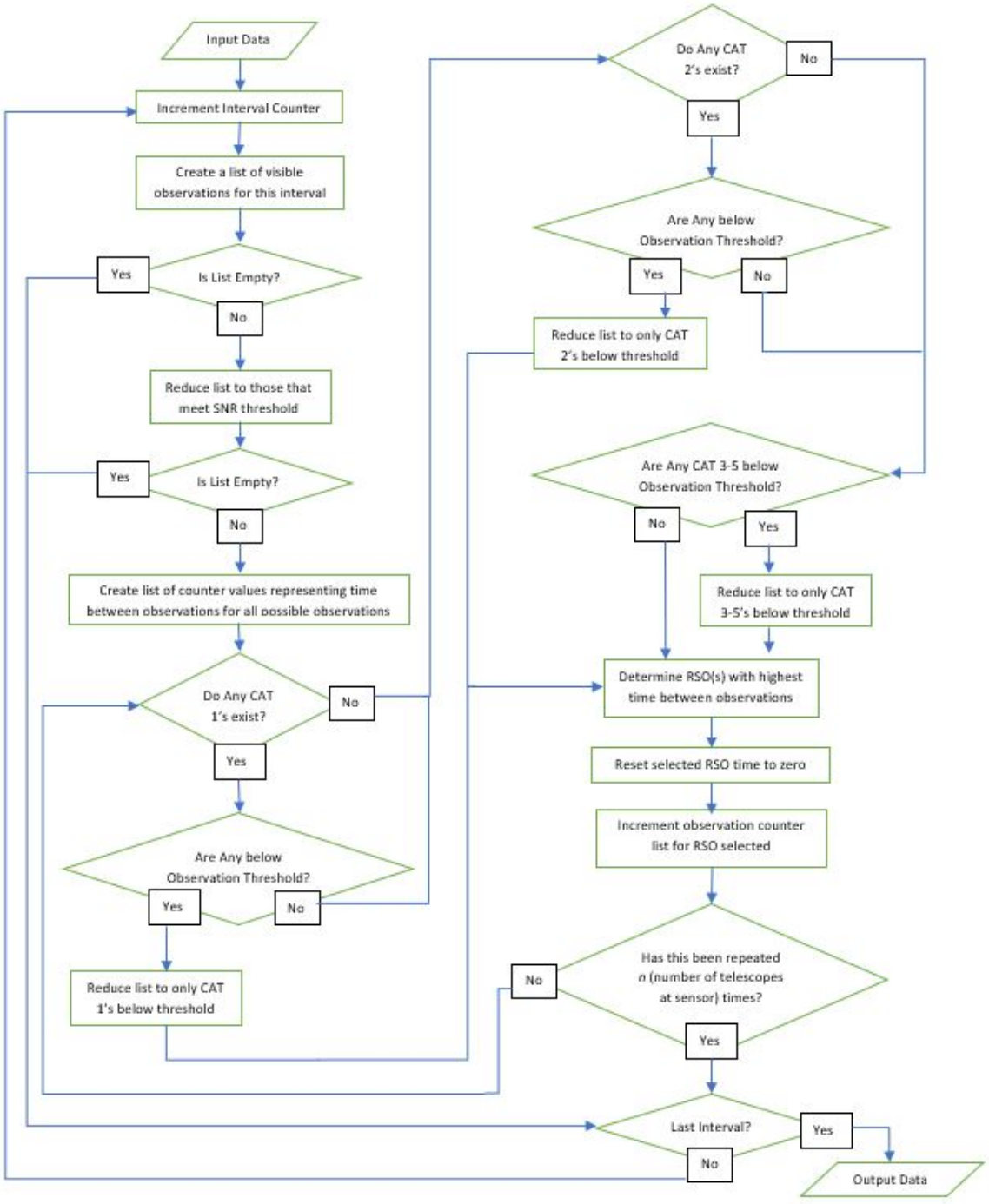

Figure 8. Relaxed SSN Scheduler Model Flow Chart

This scheduler relaxes some of the CAT 1 and CAT 2 constraints of the SSN scheduler in Section 3.6. The restrictions of the SSN Scheduler Model force a sensor to always observe a Category 1 or 2 RSO when one is in view, regardless if all available Category 1 and 2 RSO's have already met thresholds. This relaxed scheduler allows the sensor to prioritize Category 1 and 2 RSO's, but if all available CAT 1 and 2 objects have met threshold the scheduler checks beyond those categories.

Of the RSO's that are observable and meet the SNR threshold, Category 1 RSO's which have not met observation thresholds are considered, the one with the highest time between observations first. Once all observable Category 1's have met threshold or none are observable, the scheduler considers Category 2 RSO's that have not met threshold and select the highest time between observations. If all Category 1 and 2 RSO's have met thresholds or are not in view, the code considers, as one group, Categories 3-5 who have not met thresholds. If all RSO's in view have met thresholds, the scheduler selects whichever observable RSO has the highest time between observations. The flow chart is seen in Figure 8.

3.8 Relaxed SSN Scheduler Model with Added Spacing

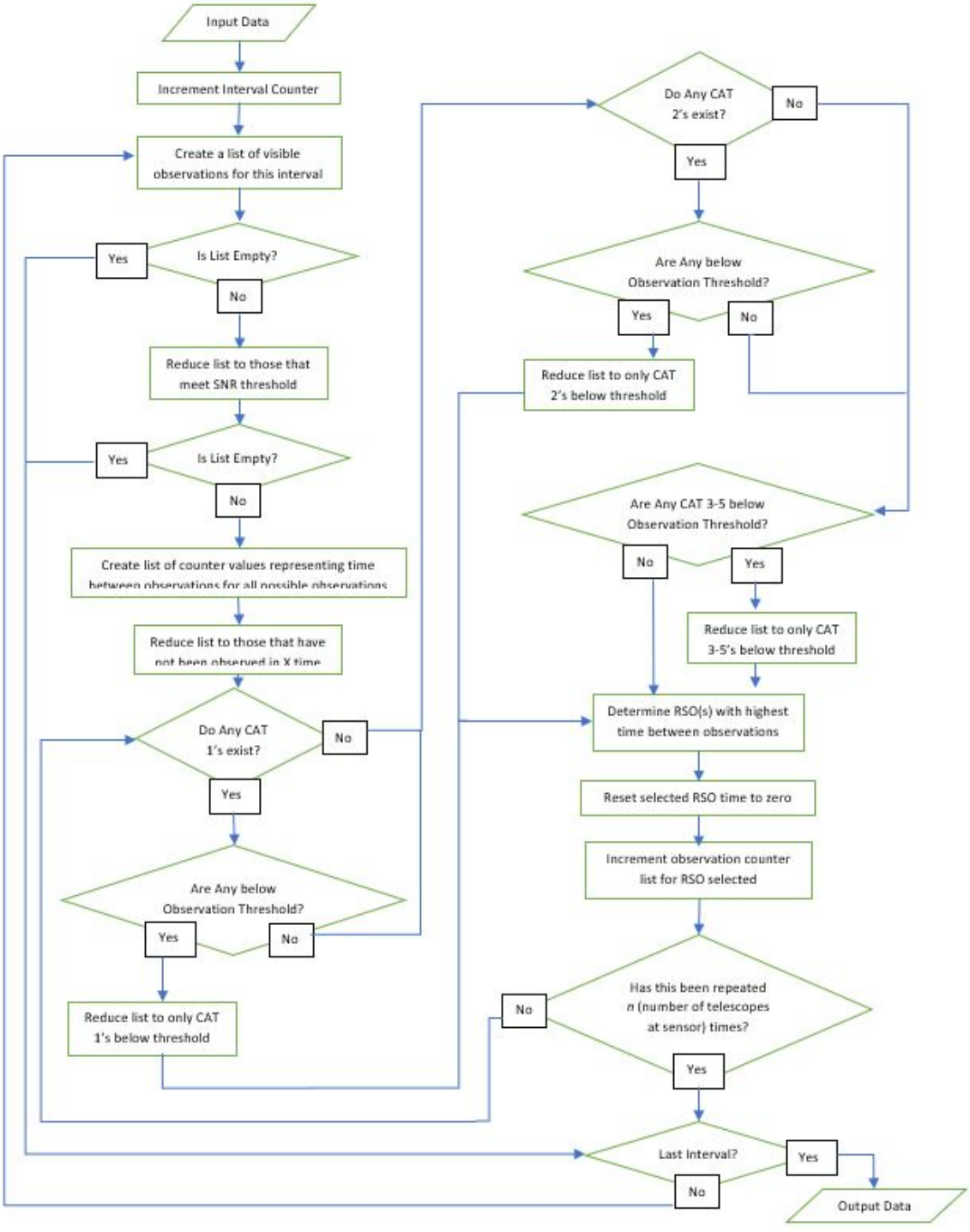

Figure 9. Relaxed SSN Scheduler Model with Added Spacing Flow Chart

This scheduler model is designed to attempt to create more adequate geometry as described by Horwood et al [24] and prevent the scheduler from observing the same RSO back to back, fulfilling the observation threshold requirements quickly, but effectively creating a lower-confidence orbit. Instead, by enforcing spacing and leaving the threshold requirement open to fill, the sensor prioritizes the RSO at a different time and forces more ideal geometry. As mentioned in Chapter II, three to four observations evenly spaced out (0.2 to 0.6 orbit periods) in the RSO's orbit are ideal to provide high-confidence orbit estimates [24]. The Relaxed SSN Scheduler Model with Added Spacing checks for CAT 1's that have not met threshold, but additionally checks to see if the same RSO has been viewed "recently." For this research, 60 interval periods, or 30 minutes is set as the gap distance between observations, which was chosen because it provides approximately a 0.3 orbital period spacing for LEO RSO's. Decision makers may adjust this number to better suit decision maker requirements. If Category 1 RSO's have not been seen in the last 30 minutes and have not met threshold, the highest among them of time between observations is selected. If all Category 1 RSO's have met threshold or they have all been seen in the last 30 minutes, or they are not observable, the scheduler repeats the process with Category 2. If the same applies to Category 2, the scheduler repeats again for Categories 3 through 5 and if all observable RSO's have met threshold or been seen in the last 30 minutes, then the scheduler picks the highest among all of them for time between observations. The flow chart is seen in Figure 9.

3.9 Binary Integer Program Scheduler Model

The Binary Integer Program Scheduler Model is a binary integer program written in python and solved using PuLP with the following model. The objective function maximizes a score value which is comprised of two main parts, the reward for viewing

an RSO and the penalty for both over viewing and under viewing RSO's. The model is constrained by the number of views the sensor may make in a single time period, the penalty variable calculations, and what the sensor can observe due to what is in range and falls within SNR ratio thresholds.

Constants:

c_i: penalty for observing object i over threshold number in one day

d_i: penalty for observing object i under threshold number in one day

g_{ij}: binary element ij in visible matrix, $g_{ij} = 1$ if visible, 0 otherwise

h_i: value of observing object i

t_i: observation threshold value of object i

Decision Variables:

x_{ij}: # of observations sensor takes of object i at time interval j in one day

y_i^+: # of observations above threshold sensor takes of object i

y_i^-: # of observations under threshold sensor takes of object i

Where:

$I := \{i | i = 1, 2, ..., 3967, 3968\}$ (The set of all RSO's)

$J := \{j | i = 1, 2, ..., 2779, 2880\}$ (The set of 30 second time intervals)

Objective Function:

$$\text{Maximize:} z = \sum_{i=1}^{3968}((\sum_{j=1}^{2880} h_i x_{ij}) - c_i y_i^+ - d_i y_i^-) \tag{20}$$

Constraints:

$$\sum_{i=1}^{3968} x_{ij} \leq 3, \forall j \in J \tag{21}$$

$$(\sum_{j=1}^{2880} x_{ij}) - y_i^+ + y_i^- = t_i, \forall i \in I \tag{22}$$

$$x_{ij} \leq g_{ij}, \forall i \in I, \forall j \in J \tag{23}$$

$$y_i^+, y_i^- \geq 0, \forall i \in I \tag{24}$$

$$x_{ij} = 0 \text{ or } 1, \forall i \in I, \forall j \in J. \tag{25}$$

The objective function, Equation 20, has two main components - the reward and penalty. The reward, $\sum_{j=1}^{2880} h_i x_{ij}$, is the sum of the number of times each RSO is observed multiplied by a value parameter based on RSO priority. For this study, the value of viewing any RSO was set to 1, as the reward for capturing or not capturing based on priority is captured in the penalty section. Thus, if 3 RSO's are observed, 3 points are rewarded. The second and third terms, $-c_i y_i^+ - d_i y_i^-$, of the objective function, reflect penalties for under viewing (not viewing an RSO enough times to meet threshold) or over viewing (observing an RSO more times than required by threshold). The model uses the penalties listed in Table 3.

Table 3. RSO Penalty Weights

Category	Under Viewing Penalty	Over Viewing Penalty
1	0.5	0.6
2	0.4	0.7
3	0.3	0.8
4	0.2	0.9
5	0.1	0.95

If the sensor views an RSO, it is rewarded with a value of 1. If that RSO was a Category 1 and it met threshold values, for example 5 observations, then it would have a value score of 5 and no penalties. If it was observed 4 times, the reward is 4, but because it was under viewed by one observation, it would receive a penalty of 0.1, bringing the score to 3.9. If it viewed the RSO six times, it would receive a reward of 6, but a penalty of 0.6 for over viewing the threshold by one. The score would then by 5.4. In this way, over viewing is more discouraged than under viewing because the penalties are higher. No matter the penalty, viewing an RSO in general provides some reward, so there is no incentive to not view any RSO's during a time interval.

The penalties are tiered in such a fashion to progressively discourage over viewing of lower categories, while minimizing penalties for under viewing these same lower categories, thus creating more incentive to view higher priority RSO's when possible.

Constraint Equation 21 ensures that only 3 RSO's are viewed by the sensor in a given time interval, representing the ability of the three telescopes at this ground station. Constraint Equation 22 forms the penalty function values y_i^+ and y_i^- where y_i^+ is assigned a value for over observing and y_i^- is assigned a value for under observing compared to threshold. For example if an RSO has a threshold of 5 and it is observed 3 times, then $y_i^- = 2$ and $y_i^+ = 0$. Constraint Equation 23 ensures that RSO's that are not in range or do not meet SNR thresholds are not selected during the given time interval. Finally, constraint Equation 24 ensures the number of penalty observations is not negative. Constraint Equation 25 ensures that during a specific time interval each RSO is either 0, not observed, or 1, observed.

The model provides a solution set which is a schedule identifying which RSO's are observed during each interval. After developing the schedule, the Binary Integer Program Scheduler Models script walks through the time intervals, resetting counters for each RSO seen at each appropriate interval. In this way, the evaluation may count the number of times an RSO has been viewed and the time between observations after the last interval for evaluation results.

3.10 Multi-Objective Scheduler Model

The Multi-Objective Scheduler Model builds upon the Binary Integer Program Scheduler Model by adding a second objective that seeks to select RSO's with high time between observations. Just as in the Binary Integer Program Scheduler Model, the first objective term is a reward for viewing an RSO, combined with a penalty for over or under viewing RSO's. However, an additional objective term is included

which provides a reward for viewing an RSO once; the reward amount based on the ratio of the time between observations for that RSO and the largest time between observations of all RSO's. The model is again constrained by the number of views the sensor may make in a single time period, the penalty variable calculations, and what the sensor can observe due to what is in range and falls within SNR ratio thresholds. Additionally, the variables for seeing the RSO are constrained by the amount of times the RSO is observed. The Multi-Objective Scheduler Model was also implemented in python and solved using PuLP with the following model.

Constants:

c_i: penalty for observing object i over threshold number in one day

d_i: penalty for observing object i under threshold number in one day

g_{ij}: binary element ij in visible matrix, $g_{ij} = 1$ if visible, 0 otherwise

h_i: value of observing object i according to meeting priority based thresholds

t_i: observation threshold value of object i

m_i: value of observing object i according to its time between observations

k: normalization constant for second objective

Decision Variables:

x_{ij}: # of observations sensor takes of object i at time interval j in one day

y_i^+: # of observations above threshold sensor takes of object i

y_i^-: # of observations under threshold sensor takes of object i

q_i: whether or not object i has been chosen for at least one observation

Where:

$I := \{i | i = 1, 2, ..., 3967, 3968\}$ (The set of all RSO's)

$J := \{j | i = 1, 2, ..., 2779, 2880\}$ (The set of 30 second time intervals)

Objective Function:

$$\text{Maximize:} z = w_1(\sum_{i=1}^{3968}((\sum_{j=1}^{2880} h_i x_{ij}) - c_i y_i^+ - d_i y_i^-)) + w_2(\sum_{i=1}^{3968} k m_i q_i) \tag{26}$$

Constraints:

$$\sum_{i=3}^{3968} x_{ij} \leq 3, \forall j \in J \tag{27}$$

$$(\sum_{j=1}^{2880} x_{ij}) - y_i^+ + y_i^- = t_i, \forall i \in I \tag{28}$$

$$x_{ij} \leq g_{ij}, \forall i \in I, \forall j \in J \tag{29}$$

$$q_i \leq \sum_{j=1}^{2880} x_{ij}, \forall i \in I \tag{30}$$

$$y_i^+, y_i^- \geq 0, \forall i \in I \tag{31}$$

$$x_{ij} = 0 \text{ or } 1, \forall i \in I, \forall j \in J \tag{32}$$

$$q_i = 0 \text{ or } 1, \forall i \in I. \tag{33}$$

Like the Binary Integer Program Scheduler Model, the first term of the objective function, Equation 26, has two main components - the reward, $\sum_{j=1}^{2880} h_i x_{ij}$, and penalty, $-c_i y_i^+ - d_i y_i^-$. The reward is the sum of the number of times each RSO is observed multiplied by the worth of seeing that RSO and the penalty for under viewing or over viewing is taken away from the score. The second term, $\sum_{i=1}^{3698} m_i q_i$, is the second objective where, if an RSO is seen at least once, then it rewards by a multiplier based on the time between observations, m_i. Both the first and second terms are multiplied by weights, w_1 and w_2, which sum to 1. Decision makers may adjust the weights to place more importance on certain objectives than on others, or may set the weights equal to each other if the objectives are equally important. For this research, Table 4 lists the different weight values examined.

Table 4. Multi-Objective Scheduler Model Objective Weights

Run	Objective 1	Objective 2
1	0.99	0.01
2	0.75	0.25
3	0.5	0.5
4	0.25	0.75
5	0.01	0.99

Constraint Equations 27, 28, 30, 31, and 32 serve the same functions as they did in the Binary Integer Program Scheduler Model. Constraint Equation 29 ensures that the variable q_i is never more than the number of observations scheduled for that RSO and constraint Equation 33 ensures q_i is binary, meaning that if there are any number of observations scheduled for an RSO, its q_i value is 1 and if it is not scheduled at all the value is 0. Constant k handles normalization of competing objective term magnitudes. Sometimes one objective has a different magnitude than other objectives [47]. In this model's case, if a sensor can see 3 objects every interval, then the first term in the objective function could potentially reach a value of 8,640, while the second term, without k, is limited to 3968. Thus a normalization constant, k is required to balance the objectives. This term is found by initially running the linear program and determining the value of both the first and second term to determine their order of magnitude, then multiplying the second term by the ratio of the first term's value over the second, and rerunning the script with this k value.

Like the Binary Integer Program Scheduler Model, after developing the schedule, the Multi-Objective Scheduler Model's script walks through the time intervals, resetting counters for each RSO seen at each appropriate interval. In this way, the evaluation may count the number of times an RSO has been viewed and the time between observations after the last interval for evaluation results.

3.11 Evaluation

Every script for each of the models contains code to count the measurement values for comparison. The script calculates the mean and maximum age between observations of all RSO's as well as in each category. The script also totals the number of observations performed on all RSO's and in each category and the number of RSO's that met target threshold values in total and by category.

3.12 Summary

Chapter III reviewed the methodology used to develop the measurement values needed to compare six different models including the baseline greedy scheduler from Stern and Wachtel. It covered how each model was developed and provided some justification on values used in the performance evaluation. Chapters IV covers results from the models in Chapter III, while Chapter V covers multi-sensor methodology and results and Chapter VI discusses conclusions of this research.

IV. Analysis

4.1 Overview

Chapter IV discusses the results produced from the models developed in Chapter III. The chapter begins with an explanation of hardware used and explains why different populations of RSO's were required. The chapter then describes initial populations for each scenario. The main portion of this chapter details test results, starting with total observation counts, counts by RSO priority, counts by percentage met by observation threshold, mean and maximum observation age. The chapter closes with script run times comparisons.

4.2 Hardware Specifications

STK data generation and Python scripts for the Base Model Greedy, the SSN Scheduler Model, the Relaxed SSN Scheduler Model, and the Relaxed SSN Scheduler model with Spacing were run on 2.5 GHz Intel Processor Computers with 64 GB RAM Running Windows 10x64. At time of writing this research, PuLP was not authorized for installation on the above hardware and was instead available on 1.7 GHz Intel Processor computer with 8 GB RAM Running Windows 10x86. PuLP, the linear program modeler, was necessary for running the Binary Integer Program Scheduler Model and the Multi-Objective Scheduler Model scripts.

4.3 Scenario Configurations

4.3.1 Scenario Dates

As mentioned in Chapter III, analysis runs were conducted on three separate days: Summer Solstice (21 June 2019), Vernal Equinox (20 March 2019), and Winter

Solstice (21 December 2018) to provide analysis of the worst case, median, and best case scenarios, respectively, for observations due to the length of nights during those dates for collection by electro-optical ground sensors.

4.3.2 Scenario RSO Population

The full available population, 3,968 RSO's, was utilized for comparison between the Base Model Greedy, the SSN Scheduler Model, the Relaxed SSN Scheduler Model, and the Relaxed SSN Scheduler model with Spacing. Due to hardware limitations using PuLP, the Binary Integer Program Scheduler Model and the Multi-Objective Scheduler Model could not be completed with the full population - the hardware available to run PuLP and Python was a 32-bit system on which PuLP and Python are restricted to 2 GB of RAM. The Binary Integer Program Scheduler Model and the Multi-Objective Scheduler Model exceed this RAM when attempting to run the scenario with greater than 190 RSO's. Therefore, additional scenarios were run for all schedulers with a population of 190 RSO's to compare equally across all of the models developed.

For each scenario, observation availability data for all 3,968 RSO's were pulled and placed in a list. The top 190 most frequently available RSO's were selected for comparison. As mentioned in Chapter III, each RSO was assigned a random priority, the breakdown of which can be seen in Table 5, and each was assigned a random initial time between observations.

Table 5. RSO Category Breakdown

Category/Suffix	3968 RSO	190 RSO
1A	6	0
1B	6	1
1C	7	1
1D	4	0
1E	11	0
2A	111	7
2B	161	8
2C	131	7
2D	150	5
2E	149	3
3A	24	1
3B	47	2
3C	41	4
3D	37	2
3E	36	4
4A	201	13
4B	208	8
4C	215	8
4D	228	11
4E	190	8
5A	395	19
5B	394	21
5C	399	26
5D	363	19
5E	454	12
1s	34	2
2s	702	30
3s	185	13
4s	1042	48
5s	2005	97

4.3.3 Multi-Objective Normalization Factor

Each of the Multi-Objective Binary Integer Program Model runs were made initially with a k value of 1. The values of each objective were totaled and used to determine the appropriate normalization factor, shown in Table 6. Additional runs

were made on the Winter Solstice to determine if adjusting the weights of each objective value would change the k value calculated, however, the k values remained constant.

Table 6. Multi-Objective Binary Integer Program Model K Values

Date/Weights	Single Sensor K Value
Summer Solstice $(w1 = 0.5, w2 = 0.5)$	15.23
Vernal Equinox $(w1 = 0.5, w2 = 0.5)$	20.75
Winter Solstice $(w1 = 0.5, w2 = 0.5)$	29.75
Winter Solstice $(w1 = 0.01, w2 = 0.99)$	29.75
Winter Solstice $(w1 = 0.25, w2 = 0.75)$	29.75
Winter Solstice $(w1 = 0.75, w2 = 0.25)$	29.75
Winter Solstice $(w1 = 0.99, w2 = 0.01)$	29.75

4.4 Total Observed

Total Observations are the number of observations the sensor was able to make on the given day. Using this metric, the distribution between priority categories can be viewed.

4.4.1 3968 RSO's

Each schedulers' results are individually analyzed in the following subsections. The main takeaways are the SSN Scheduler focuses entirely on CAT 1 or 2 RSO's without viewing any lower priority category RSO's, while both relaxed models redistribute some CAT 1 observations into CAT 2 or lower priority RSO's. The Base Model Greedy observed RSO's in proportion to the number of RSO's within that

category, however, the greedy based model is not an accurate comparison as it does not incorporate SNR thresholds.

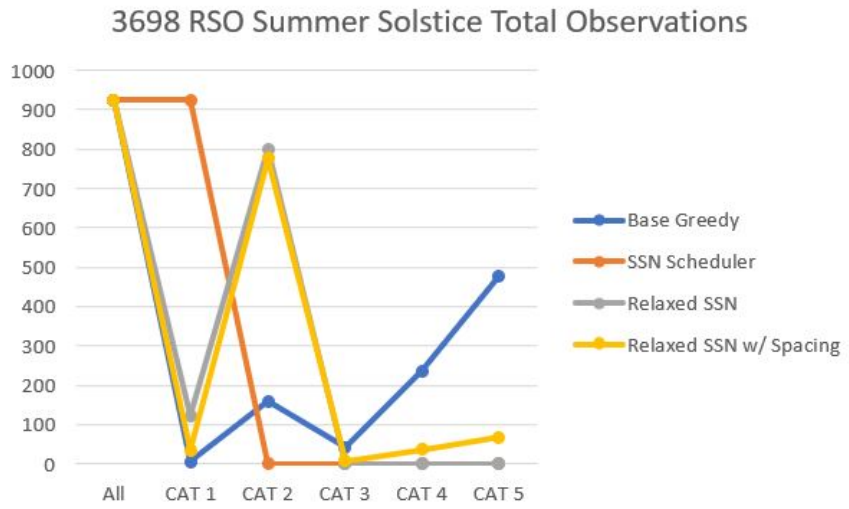

Figure 10. 3968 RSO Summer Solstice Total Observations

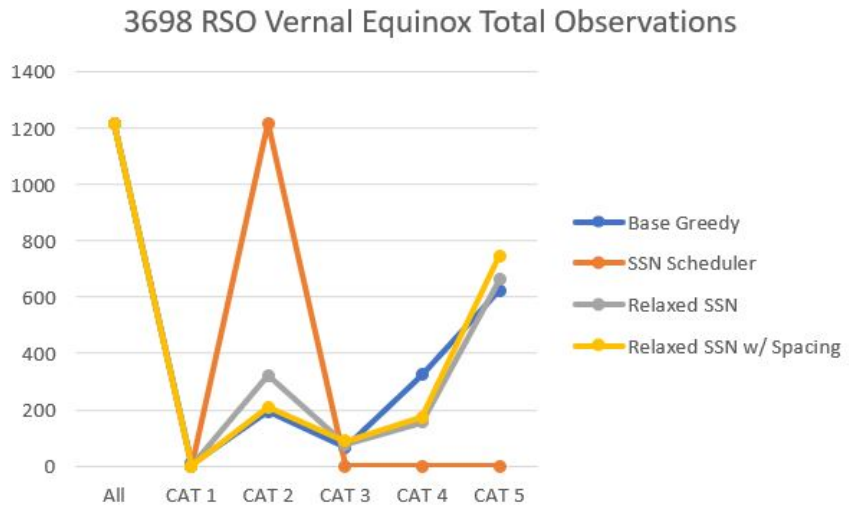

Figure 11. 3968 RSO Vernal Equinox Total Observations

4.4.1.1 Base Model Greedy

The Base Model Greedy observed RSO's fairly in proportion to the number of RSO's in each category. The lowest number of observations was in CAT 1 RSO's

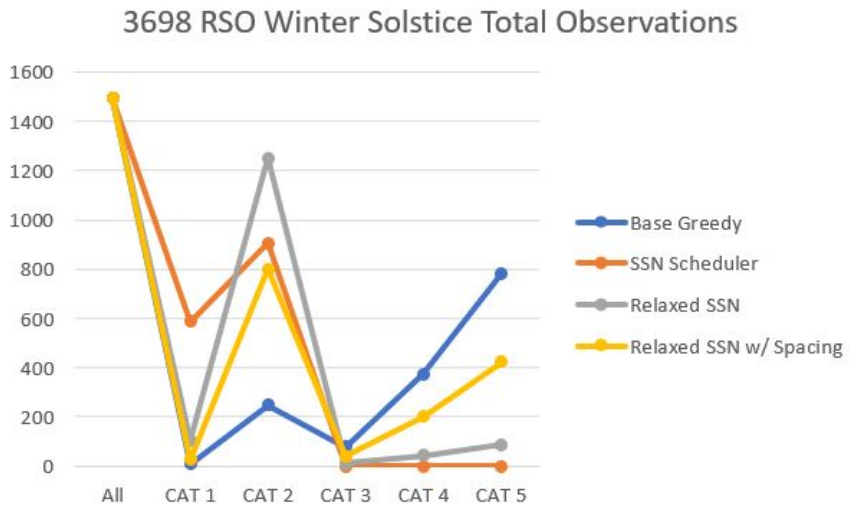

Figure 12. 3968 RSO Winter Solstice Total Observations

while the highest was in CAT 5. This model performed roughly the same on each of the three simulated days.

4.4.1.2 SSN Scheduler Model

The SSN Scheduler Model, as expected, focused exclusively on CAT 1 and 2 RSO's. On the Summer Solstice, the model only viewed CAT 1's, while on the Vernal Equinox, it only viewed CAT 2's. The lack of CAT 1's on the Vernal Equinox is due to those RSO's not having a sufficient SNR value on that date, therefore the modeler went to CAT 2's. On the Winter Solstice, the model viewed both CAT 1's ad 2's but nothing else. This result closely matches real world trends where sensor capacity tends to absorb viewing CAT 1 and CAT 2 RSO's, preventing adequate viewing of lower priority objects.

4.4.1.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model observed RSO's in every category, with a larger emphasis on CAT 2's due to the higher number within that population. It observed

significantly less CAT 1's and more CAT 2's on the Summer Solstice, but did not view any CAT 3-5 on this day. On the Vernal Equinox CAT 3 and 4's were viewed and the number of CAT 5's were higher than any other category. On the Winter Solstice, the emphasis area was in CAT 2's, but this day the model observed RSO's in all categories. This is a positive result as the model is maintaining assurance that visible CAT 1 and CAT 2 objects are viewed to their observation threshold. Relaxing the "observe every pass" restriction in 2004 SD 505-1 enables a shift to lower priorities.

4.4.1.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model performs similarly to the previous model, but with additional emphasis in CAT 3-5's. On the Summer and Winter Solstice, it observed less CAT 1's than either of the two previous models, but was able to observe some RSO's in every category. On the Vernal Equinox, like the Relaxed SSN Scheduler Model a higher number of CAT 3-5 were seen, CAT 5's being the largest of all categories seen. This performs as expected - the scheduler focuses less on CAT 1 and 2's, like the Relaxed SSN Scheduler, but because of the spacing consideration on CAT 1 and 2's, the scheduler cannot revisit those same RSO's as frequently. Thus, additional CAT 3-5's can be observed.

4.4.1.5 Conclusion

The SSN Scheduler truly focused on CAT 1 and 2 RSO's to the point where all other categories were ignored on each scenario day. Both relaxed models observed less CAT 1's but were generally able to view more CAT 2's, with exception to the Vernal Equinox, and view more CAT 3-5's. The Relaxed SSN Scheduler with Spacing was able to further spread observations to CAT 3-5's more so than the Relaxed SSN Scheduler. The Base Model Greedy observed RSO's in proportion to the population.

The Total Observation metric identifies how the SSN Scheduler truly limits focus to CAT 1 and 2 RSO's. At the same time, the novel schedulers are shown to spread observations to lower priority categories.

4.4.2 190 RSO's

Detailed analysis of each scheduler for the 190 RSO scenarios is included in the following subsections. Overarching trends are that in a smaller population, almost every model except the SSN Scheduler observes RSO's in a similar fashion to the Base Model Greedy - proportional to the number of RSO's in each categories. The shift from single to Multi-Objective did not perpetuate any notable differences, while the SSN Scheduler did not view anything except CAT 1 and 2's.

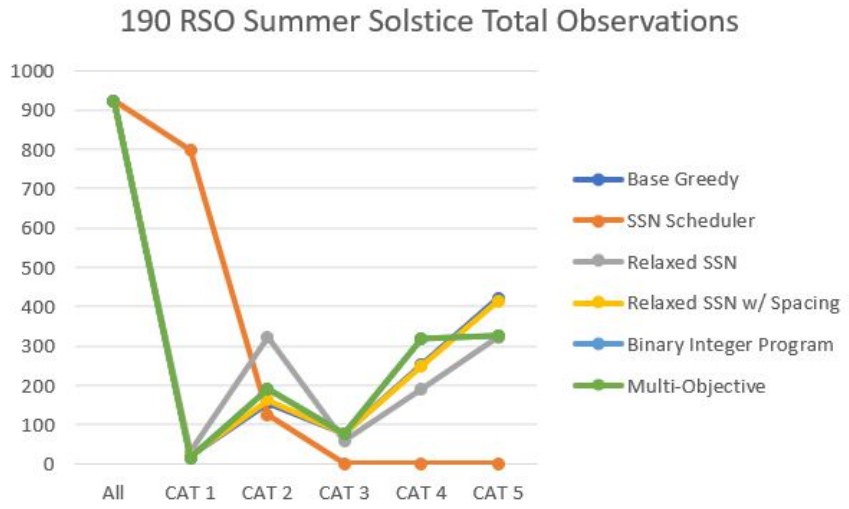

Figure 13. 190 RSO Summer Solstice Total Observations

4.4.2.1 Base Model Greedy

Like in the 3968 RSO scenarios, this model generally made observations in proportion to the population. CAT 5 RSO observations were the highest category viewed.

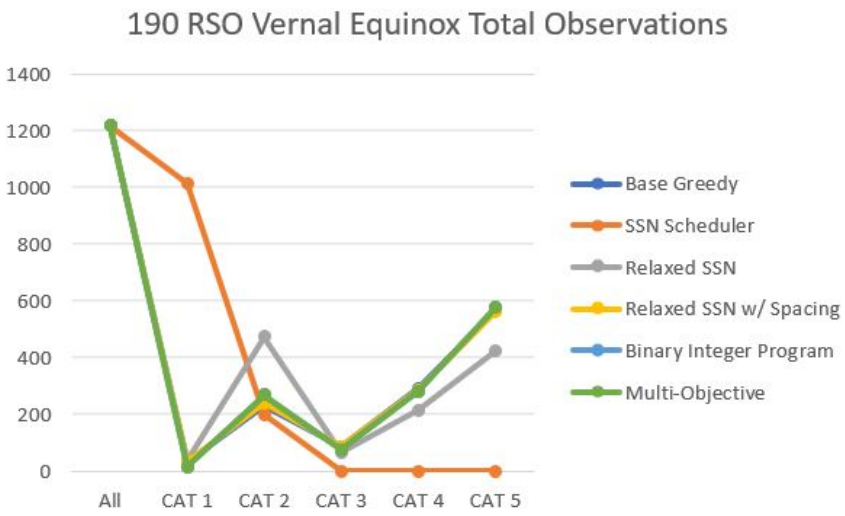

Figure 14. 190 RSO Vernal Equinox Total Observations

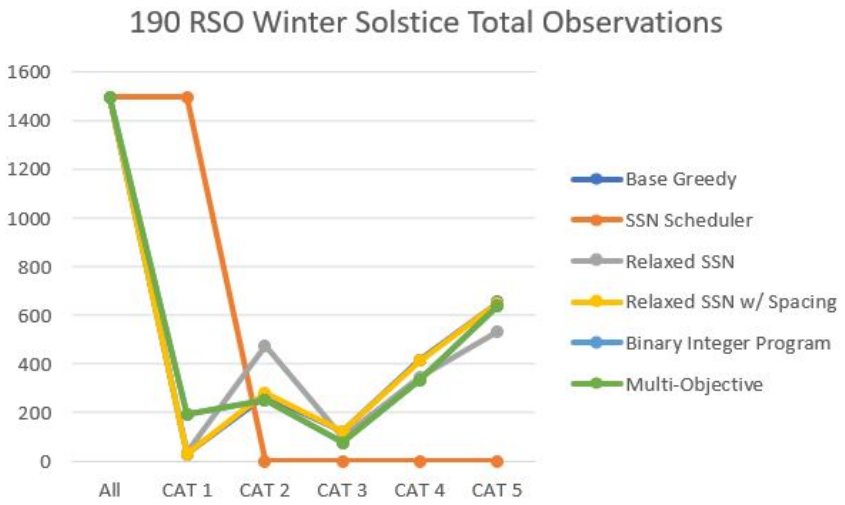

Figure 15. 190 RSO Winter Solstice Total Observations

4.4.2.2 SSN Scheduler Model

The SSN Scheduler Model, like in the larger data set, primarily focused on CAT 1 and 2 RSO's but managed to view both categories on the Summer Solstice and Vernal Equinox, but focused singularly on CAT 1's on the Winter Solstice.

4.4.2.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model views at least double the number of CAT 2's than the SSN Scheduler, while viewing a significant amount of CAT 3-5's more than the previous model on each day. The result numbers were very similar to the Base Model Greedy.

4.4.2.4 Relaxed SSN Scheduler with Spacing Model

The second relaxed model performed very similarly to the first Relaxed SSN Scheduler Model, but observed less CAT 2's and observed more lower priority RSO's, again resembling the Base Model Greedy.

4.4.2.5 Binary Integer Program Model

The integer program model viewed less CAT 1's than all previous models, while viewing approximately the same number of CAT 2's as the Relaxed SSN Scheduler Model with Spacing each day. On the Winter Solstice, the integer program model viewed more CAT 1's than the other models except for the SSN Scheduler Model. This shows the Binary Integer Program has the capability to spread observations to lower priority categories and not focus unnecessarily on CAT 1's. The integer program found more value in observing CAT 2 RSO's and, as shown later, the integer program is able to fulfill more thresholds because of the ability to spread observations to lower priorities.

4.4.2.6 Multi-Objective Binary Integer Program Model

The Multi-Objective Model with equally weighed objectives performed exactly the same as the Binary Integer Program Model. The same performance indicates the

addition of a second objective does not impact the solution in a way to drastically change the number observations made in each category.

4.4.2.7 Conclusion

Every model except the SSN Scheduler made observations similar to the proportion of RSO's in each category, like the Base Model Scheduler usually does. Both relaxed models and both integer program models observed a larger number of CAT 3-5 RSO's, while the SSN Scheduler did not view any. The Multi-Objective model did not show any differences. Both single and multi-objective integer programs showed a spread of observations to lower priority RSO's, indicating more lower priority RSO's have the opportunity to meet threshold.

4.5 Total RSO's Meeting Observation Threshold

Each RSO has an associated suffix code A-E which determines its observation threshold in accordance with 2004 SD 505-1. This section captures both the count of RSO's meeting threshold by category, as well as the percentage of RSO population meeting threshold by category.

4.5.1 3968 RSO's

Each schedulers results are detailed in the following subsections. Overarching trends are that the SSN Scheduler Model performs the worst in meeting threshold requirements, while the Relaxed Constraint and Relaxed SSN Scheduler with Spacing models allow a significantly larger amount of RSO's to meet threshold requirements. Novel schedulers are shown to improve CAT 2 threshold numbers by up to 15% with a 3% increase in all RSO's. The Base Model Greedy performs the best in terms of most RSO's meeting threshold overall, however it is again important to recall that

the base greedy code, mirroring prior research, does not include SNR limitations.

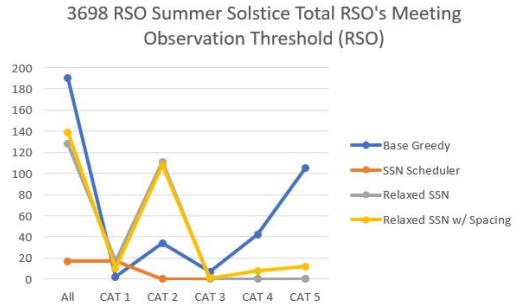

Figure 16. 3968 RSO Summer Solstice Total RSO's Meeting Observation Threshold

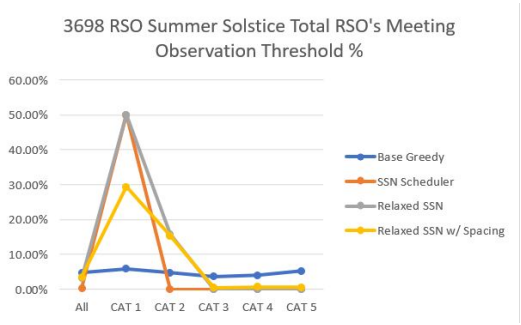

Figure 17. 3968 RSO Summer Solstice Total RSO's Meeting Observation Threshold %

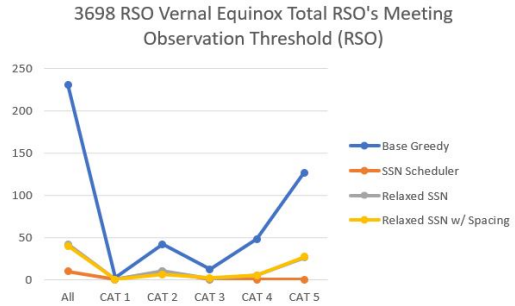

Figure 18. 3968 RSO Vernal Equinox Total RSO's Meeting Observation Threshold

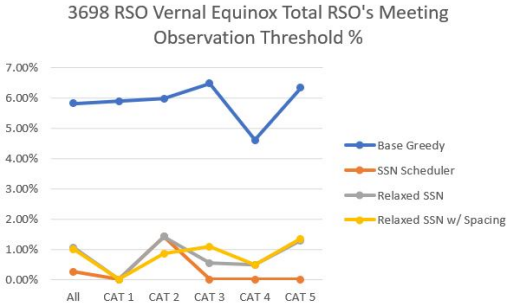

Figure 19. 3968 RSO Vernal Equinox Total RSO's Meeting Observation Threshold %

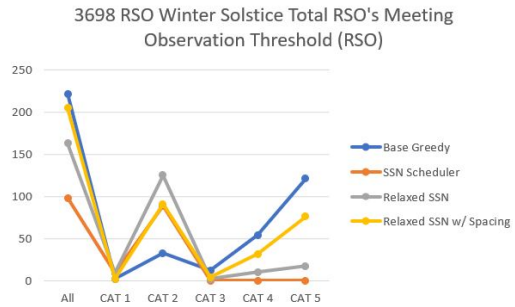

Figure 20. 3968 RSO Winter Solstice Total RSO's Meeting Observation Threshold

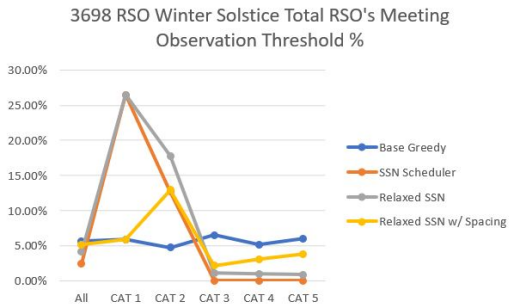

Figure 21. 3968 RSO Winter Solstice Total RSO's Meeting Observation Threshold %

4.5.1.1 Base Model Greedy

The Base Model Greedy had the largest number of RSO's meeting threshold. These results are proportional to the number of RSO's in each category. This corresponds to a flat percentage of RSO's meeting threshold priority at around 5%. However, recall that the base greedy model, mirroring prior research, does not include an SNR cutoff.

4.5.1.2 SSN Scheduler Model

The SSN Scheduler Model performed the worst out of all schedulers in meeting thresholds. Because the scheduler focused on CAT 1 and 2 RSO's, only a select few RSO's actually made threshold, on Vernal Equinox only 10 CAT 1's made threshold. On the Winter Solstice, a larger number of CAT 2's were able to meet threshold, likely due to no CAT 1's being visible at times. This model manages to meet 25-50% of the population of CAT 1's on both solstices, but like all other schedulers except the base model cannot view any on Vernal Equinox due to low SNR. This indicates the SSN Scheduler Model only meets threshold of a select number of RSO's, which are CAT 1 or CAT 2's, and makes additional observations beyond threshold requirements into those same RSO's. Meeting an RSO's threshold already provides high-confidence estimates, thus over observing an RSO would be inefficient where other RSO's of lower priority could be observed and have the opportunity to meet their thresholds.

4.5.1.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model shows a significant improvement compared to the SSN Scheduler. The model meets the same amount of CAT 1's, and allows the same amount or more RSO's to meet threshold in all other categories. On the Summer Solstice the model did not view any CAT 3-5, likely due to the large number

of CAT 2's visible. Improvement indicates relaxing the constraints of CAT 1 and 2 observation allows additional RSO's of lower priority to meet observation thresholds. At the same time, doing so does not sacrifice thresholds of CAT 1's or 2's significantly, allowing the scheduler to provide a more efficient and beneficial solution.

4.5.1.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model performs very similarly to the Relaxed SSN Scheduler Model, outperforming that model on both solstices. The total number meeting threshold is only slightly under that of the Relaxed SSN Scheduler Model, viewing a slightly lower number of CAT 2's that day. In general, less CAT 1 and CAT 2 RSO's meet threshold while more CAT 3-5 do. The slight drop in threshold numbers is expected as an additional constraint is placed upon the Relaxed SSN Scheduler Model. Adding an additional constraint for spacing minimally affects the benefits gained in relaxing the CAT 1 and CAT 2 constraints, but provides the potential for better geometry of observations.

4.5.1.5 Conclusion

In terms of maximizing the number of RSO's meeting threshold requirements, the SSN Scheduler Model performs the worst by far. The best model in this scenario is the Relaxed SSN Scheduler with Spacing Model, which meets less CAT 1 and 2 thresholds to fulfill a larger amount of CAT 3-5. However, the Relaxed SSN Scheduler Model manages to meet the same thresholds as the SSN Scheduler or do much better in all categories. Novel schedulers are shown to improve CAT 2 threshold numbers by up to 15% with a 3% increase in all RSO's. The increase in threshold numbers of the both the Relaxed SSN Scheduler Model and the Relaxed SSN Scheduler Model with Spacing is expected as the constraints are reduced on CAT 1 and CAT 2 obser-

vations. At the same time, the model with spacing having met less thresholds is also expected as an additional constraint is placed, but should provide better geometry for observations.

4.5.2 190 RSO's

Detailed analysis of each scheduler for the 190 RSO scenarios is included in the following subsections. Overarching trends are that the SSN Scheduler only allows a small number of CAT 1 and CAT 2 RSO's to meet threshold, which is bested every time by almost every other model. At least a 20% improvement is seen in all models in overall threshold numbers and at least a 20% increase in CAT 3-5's is seen in all novel schedulers. The integer programs perform the best, allowing at least six times the number of RSO's to meet threshold than the SSN Scheduler Model.

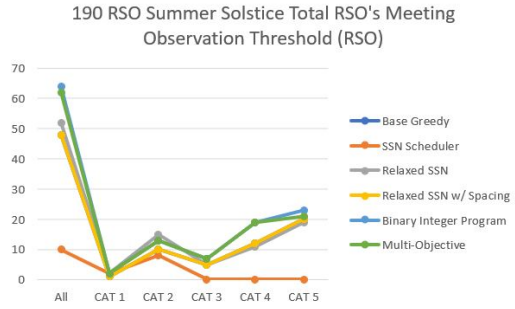

Figure 22. 190 RSO Summer Solstice Total RSO's Meeting Observation Threshold

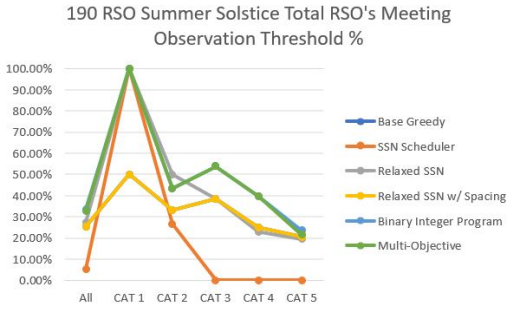

Figure 23. 190 RSO Summer Solstice Total RSO's Meeting Observation Threshold %

4.5.2.1 Base Model Greedy

In this smaller population, the Base Model Greedy follows established trends by allowing RSO's to meet threshold in proportion to the number in each category. Performance of this model is mid-range compared to the other models.

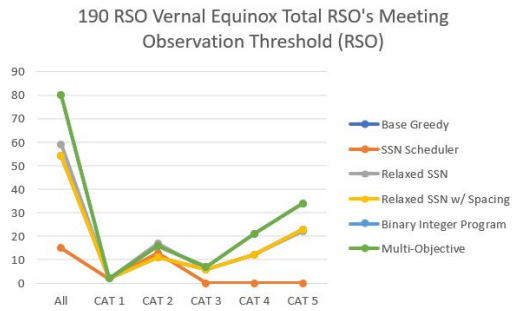

Figure 24. 190 RSO Vernal Equinox Total RSO's Meeting Observation Threshold

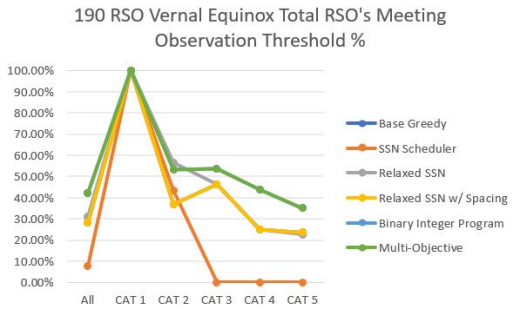

Figure 25. 190 RSO Vernal Equinox Total RSO's Meeting Observation Threshold %

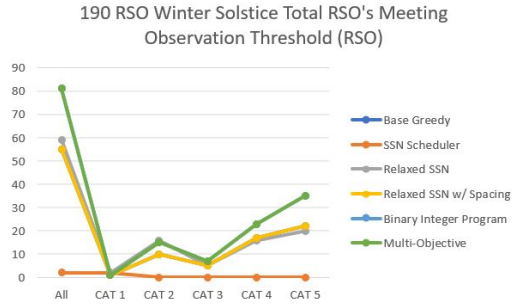

Figure 26. 190 RSO Winter Solstice Total RSO's Meeting Observation Threshold

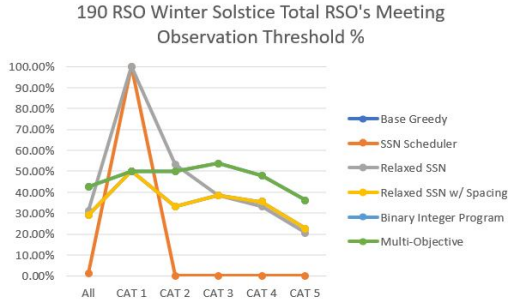

Figure 27. 190 RSO Winter Solstice Total RSO's Meeting Observation Threshold %

4.5.2.2 SSN Scheduler Model

Once again, the SSN Scheduler Model performs the worst, maximizing the number of CAT 1 and CAT 2 RSO's meeting threshold. However, because these higher priority RSO's are in view, the scheduler often does not attempt to view any other category. Yet, even with this heavy emphasis on CAT 1's and 2's, the scheduler is only meeting observation thresholds at the same level of other schedulers.

4.5.2.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model allows the most number of CAT 2 RSO's to meet threshold, while maintaining the same number of CAT 1's as other models. The rest of the categories' performance is mid-range. The performance of this scheduler is

expected as the constraints on CAT 1 and 2's is relaxed. Because the scheduler meets the same thresholds of CAT 1's, this indicates the scheduler performs more efficiently than the SSN Scheduler Model and the fact CAT 2's meeting threshold is highest among all models indicates this scheduler performs best for improving performance of high priority RSO's.

4.5.2.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model has almost identical results to the Base Model Greedy, whose performance is mid-range, better than the SSN Scheduler but trumped by the integer programs. The results are slightly below the Relaxed SSN Scheduler Model which is expected because of the additional constraints. Although CAT thresholds met drops 50%, recall in this scenario there are only two CAT 1's, thus only one CAT 1 was sacrificed for the improved geometry. On a larger scenario population, a smaller drop is expected. These results positively show that the geometry can be improved with minimal effect on threshold performance.

4.5.2.5 Binary Integer Program Model

The Binary Integer Program Scheduler Model performs the best in terms of allowing the most RSO's to meet threshold in this population size. The number meeting threshold is higher in every category than any other modeler, except for CAT 2's which are slightly less than the results with the Relaxed SSN Scheduler Model. This is expected as the Binary Integer Program Model's goal is to get the most RSO's to meet threshold, especially that of higher priorities. The Relaxed SSN Scheduler Model was expected to be the best performer for CAT 2's because it relaxed constraints on CAT 1 and 2's, but still placed the most focus on those higher priority categories.

4.5.2.6 Multi-Objective Binary Integer Program Model

Once again, the multi-objective model performs almost identically to the Binary Integer scheduler, only slightly lower during Summer Solstice. This indicates adding an additional objective minimally affects the number of thresholds met, but allows performance to be increased in other areas.

4.5.2.7 Conclusion

The Binary Integer scheduler allows the most RSO's to meet observation thresholds. The SSN Scheduler Model performs the worst limiting only CAT 1 and CAT 2 priorities to meet threshold. At least a 20% improvement is seen in all models in overall threshold numbers and at least a 20% increase in CAT 3-5's is seen in all novel schedulers. The results indicate the integer program models perform the best at creating schedules which will fulfill the most thresholds, especially that of high priority RSO's. They are expected to be more efficient in meeting observation thresholds than the relaxed models, though the Relaxed SSN Scheduler Model is expected to perform best in CAT 2's.

4.6 Mean Age

The Mean Age metric is the average time between observations for all RSO's in a category. As the time until between RSO observations increases the uncertainty of the orbit approximation also increases. Reducing the mean age of all the RSO's indicates how well the scheduler reduces uncertainty in orbits across the RSO population.

4.6.1 3968 RSO's

Each schedulers results are detailed in the following subsections. Overarching trends are that the SSN Scheduler performs the worst only reducing CAT 1 mean

age, while other schedulers do the same and more. Up to a 36% decrease in mean age of CAT 2's and a 13% decrease in CAT 3-5 mean age is viewed in novel schedulers. The best performing scheduler at reducing mean age is the Relaxed SSN Scheduler Model with Spacing, while the Base Model Greedy keeps mean age low and level. However, recall the Base Greedy Model, which captures prior research methodologies, does not include an SNR filter and thus is expected to have higher level performance.

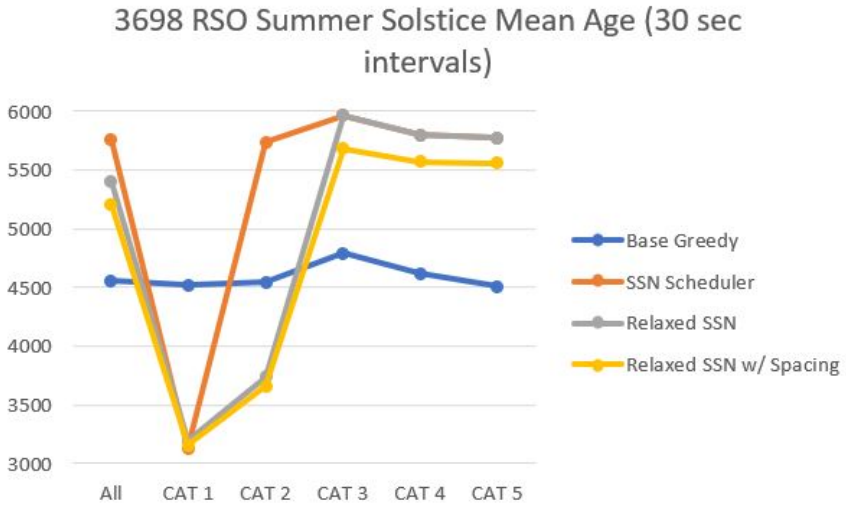

Figure 28. 3968 RSO Summer Solstice Mean Age

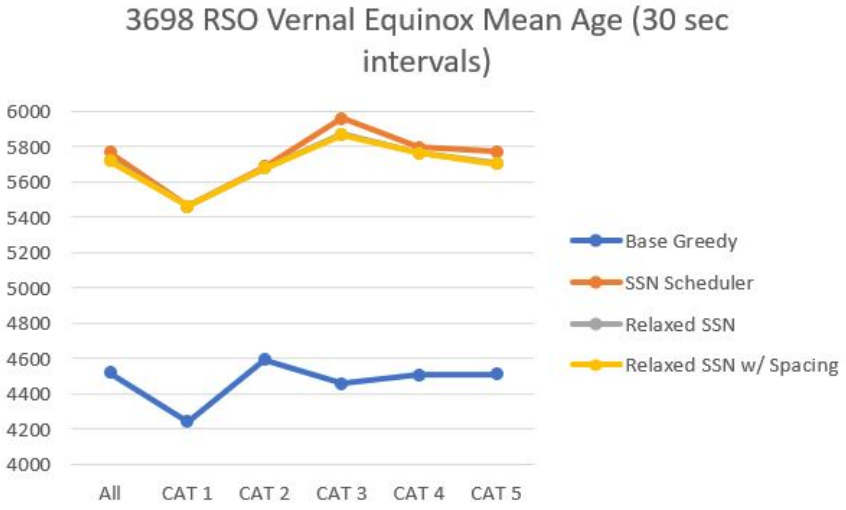

Figure 29. 3968 RSO Vernal Equinox Mean Age

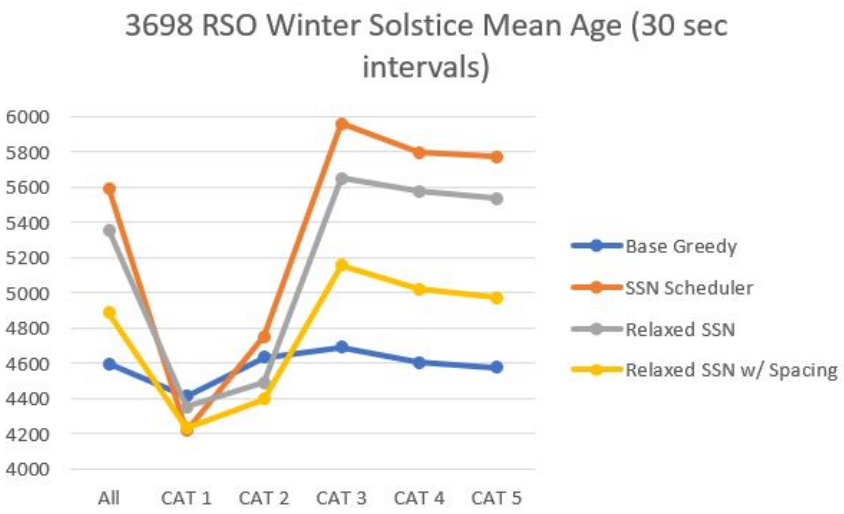

Figure 30. 3968 RSO Winter Solstice Mean Age

4.6.1.1 Base Model Greedy

The Base Greedy Model maintained a level mean age for all categories around 4500 intervals. The highest mean age was at Summer Solstice with 4789 intervals and the lowest was during Vernal Equinox at 4242 intervals. Again, note that the Base Greedy Model, which captures prior research methodologies, does not include an SNR filter and thus is expected to perform better than other models as it has greater flexibility in available RSO's.

4.6.1.2 SSN Scheduler Model

The SSN Scheduler in general performs the worst with mean age. It focuses on bringing down the mean age of CAT 1 RSO's, however, it is matched by every other scheduler at this category, at the same time, the other categories mean ages are all among the worst across all schedulers.

4.6.1.3 Relaxed SSN Scheduler Model

The Relaxed Constraint performs slightly better than the SSN Scheduler. It nearly matches the SSN Scheduler in CAT 1, while besting CAT mean age by 2000 intervals during the Summer Solstice. During Vernal Equinox, all of the models are fairly on par with each other, while on the Winter Solstice, the relaxed model performs worse in CAT 1's by 100 intervals but better in almost every other category by about 300 intervals.

4.6.1.4 Relaxed SSN Scheduler with Spacing Model

The relaxed model with spacing outperforms the Relaxed SSN Scheduler Model. It has lower mean age on each day in every category than the relaxed model. This is a positive result for this scheduler model. The goal of adding required spacing to this model to distribute observations more equitably throughout each pass, thus it is expected that mean age should drop.

4.6.1.5 Conclusion

The SSN Scheduler performs the worst at reducing mean age. It does adequately for CAT 1, but other schedulers match that and reduce other categories at the same time. Up to a 36% decrease in mean age of CAT 2's and a 13% decrease in CAT 3-5 mean age is viewed in novel schedulers. The best overall scheduler at reducing mean age is the Relaxed SSN Scheduler Model with Spacing.

4.6.2 190 RSO's

Detailed analysis of each scheduler for the 190 RSO scenarios is included in the following subsections. Overarching trends are that the SSN Scheduler is outperformed by almost all novel schedulers. Up to a 33% improvement is seen in mean age of all

RSO's while up to a 16% improvement in CAT 1 and a 19% improvement or greater improvement in CAT 2-5's mean age in novel schedulers. The Relaxed SSN Scheduler with Spacing Model in particular performs slightly better than the rest.

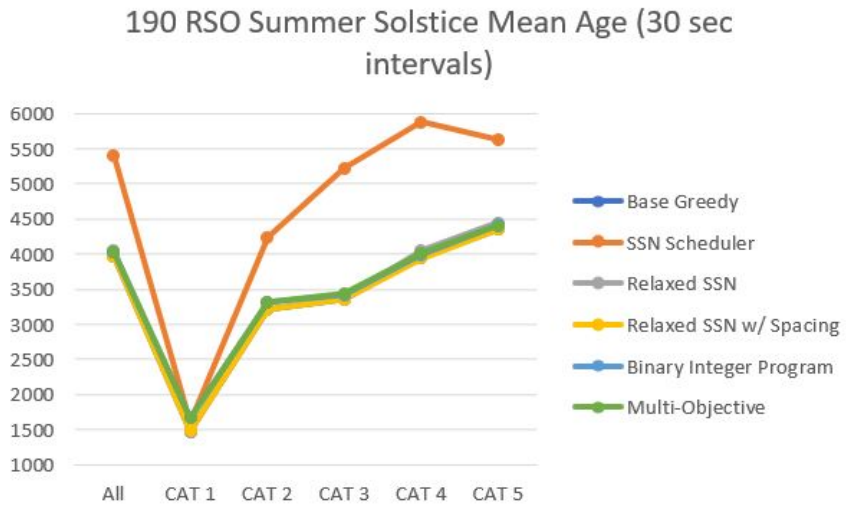

Figure 31. 190 RSO Summer Solstice Mean Age

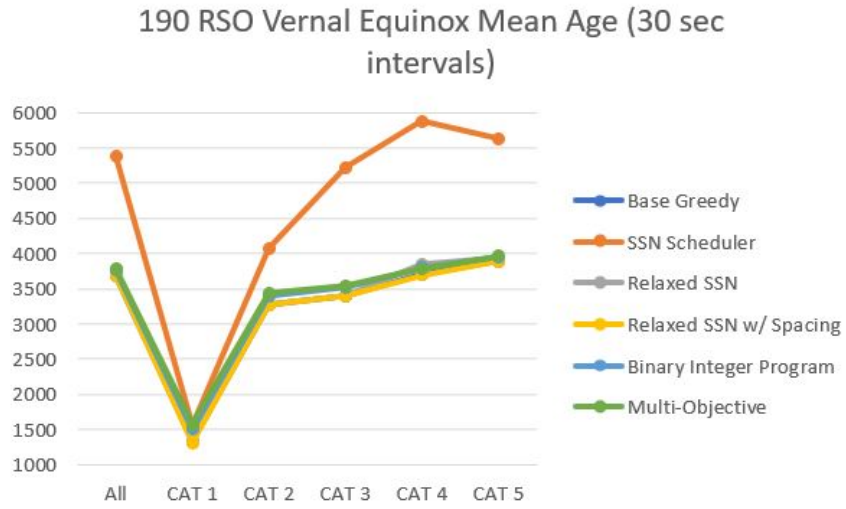

Figure 32. 190 RSO Vernal Equinox Mean Age

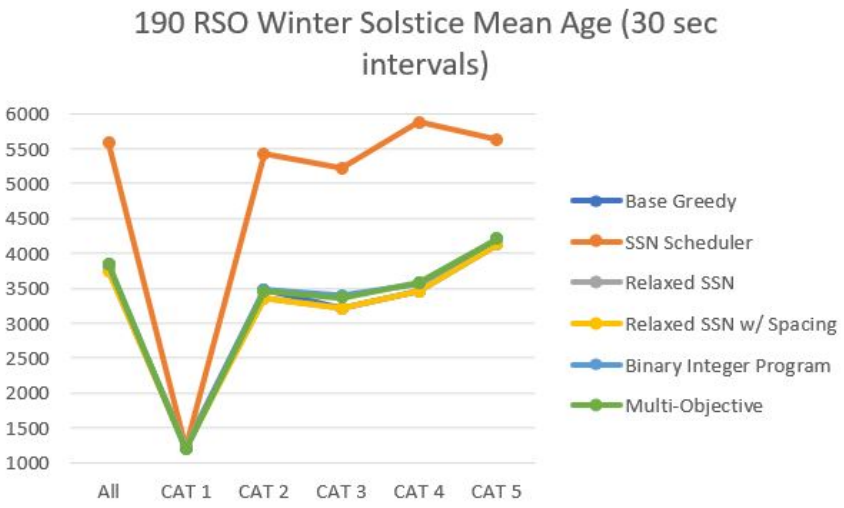

Figure 33. 190 RSO Winter Solstice Mean Age

4.6.2.1 Base Model Greedy

The Base Model Greedy performs on par with all other models except the SSN Scheduler. It reduces CAT 1 mean age primarily because there are only 2 CAT 1's in the smaller population.

4.6.2.2 SSN Scheduler Model

The SSN Scheduler performs the worst at reducing mean age. It is the only scheduler using this smaller population of RSO's that does not reduce mean age of CAT 2-5 RSO's versus baseline ages, indicating, as seen in prior sections, that this scheduler is not observing these categories.

4.6.2.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler models is one of the better performing models on par with everything but the SSN Scheduler.

4.6.2.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model slightly performs better at reducing mean age than all other schedulers. The one exception to this is CAT 1 mean age on the Winter Solstice, for which the integer programming and multi-objective schedulers were superior.

4.6.2.5 Binary Integer Program Model

The Binary Integer Program Model performs on par with most other schedulers but is slightly behind the second relaxed model until Winter Solstice in CAT 1.

4.6.2.6 Multi-Objective Binary Integer Program Model

The multi-objective model is nearly exactly the same as the Binary Integer Programming Model with only small differences and is on par with the rest.

4.6.2.7 Conclusion

Almost every model in the 190 RSO population reduces the mean age by around the same amount except for the SSN Scheduler model. Up to a 33% improvement is seen in mean age of all RSO's while up to a 16% improvement in CAT 1 and a 19% improvement or greater improvement in CAT 2-5's mean age in novel schedulers. The Relaxed SSN Scheduler with Spacing Model has a slight edge in most areas.

4.7 Max Age

Maximum age is the largest time between observations for all the RSO's in the population or a category. It shows the worst case scenario for an RSO and shows how the scheduler performs at dealing with reducing the worst targets.

4.7.1 3968 RSO's

Each schedulers results are detailed in the following subsections. Overarching trends are that there is small difference in max age when running a single sensor on a large population. Up to a 0.56% improvement in maximum age within CAT 3's is viewed in novel schedulers. The Base Model Greedy is able to make an impact because it is not constrained by SNR, however, the Relaxed SSN Scheduler with Spacing Model comes close and slightly outperforms the rest.

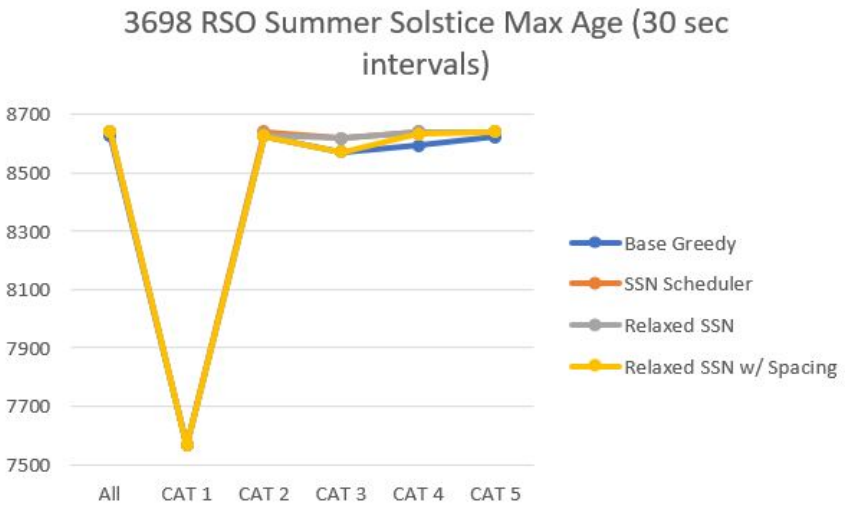

Figure 34. 3968 RSO Summer Solstice Max Age

4.7.1.1 Base Model Greedy

The Base Model Greedy generally is on par or slightly better than all other models in reducing the max age of the large population. The only area that differs majorly is CAT 4's where this scheduler reduces max age further than the rest. During the Vernal Equinox it outperforms the rest because it is not constrained by SNR ratio and can freely observe those RSO's.

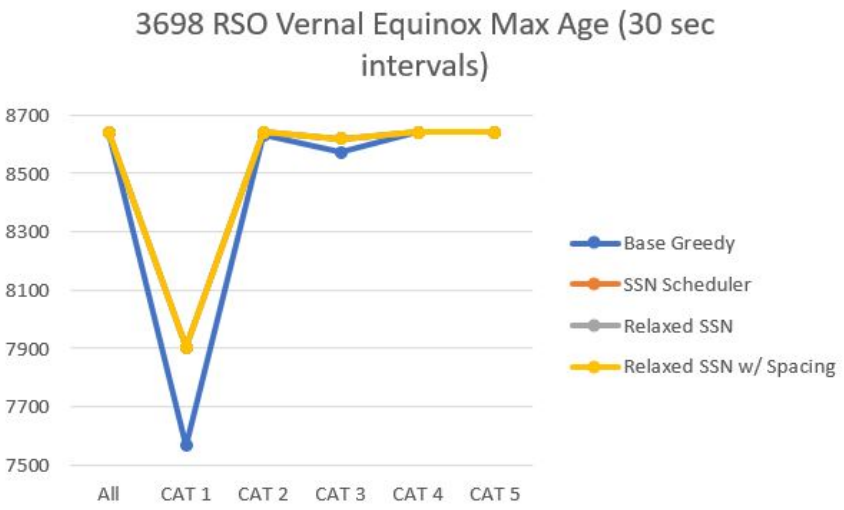

Figure 35. 3968 RSO Vernal Equinox Max Age

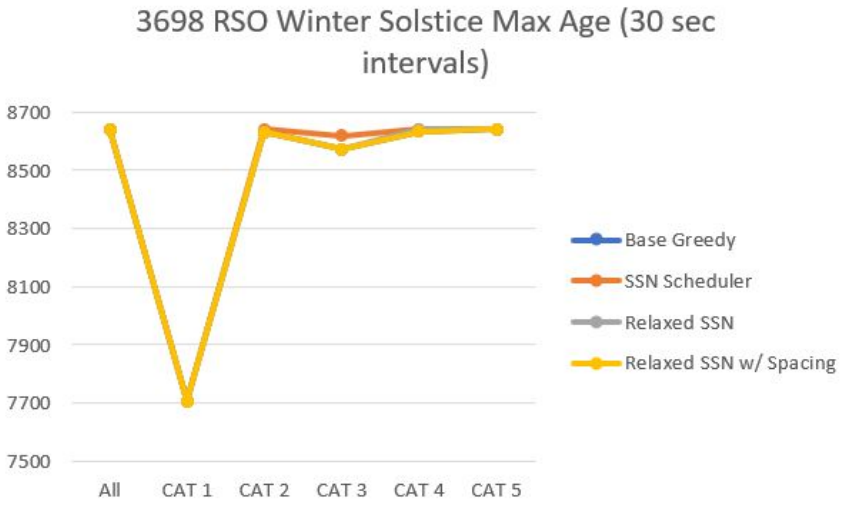

Figure 36. 3968 RSO Winter Solstice Max Age

4.7.1.2 SSN Scheduler Model

The SSN Scheduler in the large population is about on par with the other schedulers except for CAT 3 where it is among the worst at reducing max age.

4.7.1.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model performs nearly the same as the SSN Scheduler Model, performing among the worst in CAT 3 areas.

4.7.1.4 Relaxed SSN Scheduler with Spacing Model

The relaxed model with spacing performs nearly on par with the greedy model on both solstices. It outperforms the SSN Network in CAT 3's on the Winter Solstice.

4.7.1.5 Conclusion

There is very little change in max age in the large population, however, the Base Model Greedy Scheduler is able to outperform the rest because it is not constrained by SNR and the Relaxed SSN Scheduler with Spacing Model comes close behind in improving max age. Up to a 0.56% improvement in maximum age within CAT 3's is viewed in novel schedulers. Considering that the Relaxed SSN Scheduler with Spacing Model is constrained by SNR, this is a significant feat.

4.7.2 190 RSO's

Detailed analysis of each scheduler for the 190 RSO scenarios is included in the following subsections. Overarching trends are that each of the schedulers have very small differences in the amount they reduce max age by, though up to a 27% decrease of max age of CAT 1's, up to 1% improvement in CAT 2's, and up to 4% improvement in CAT 5's is seen in novel schedulers. The Relaxed SSN Scheduler mode, very slightly outperforms the rest in CAT 1's, while the SSN Scheduler and Relaxed SSN Scheduler Model perform slightly worse in CAT 5.

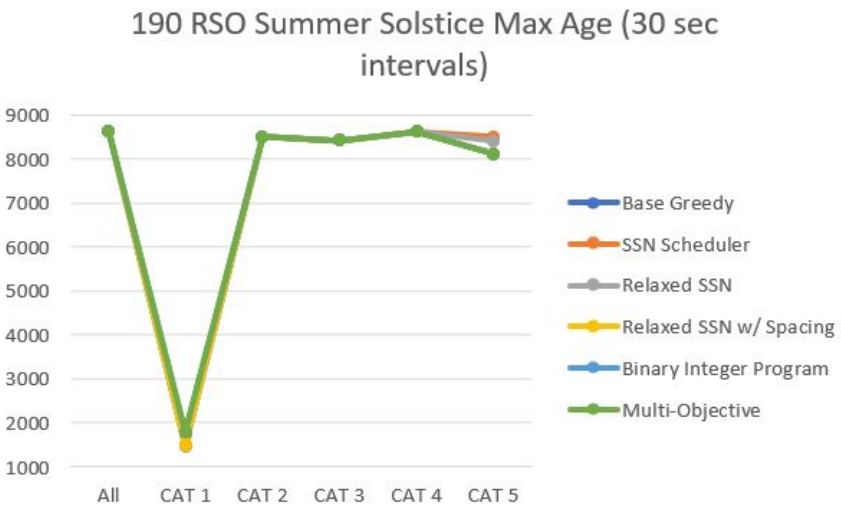

Figure 37. 190 RSO Summer Solstice Max Age

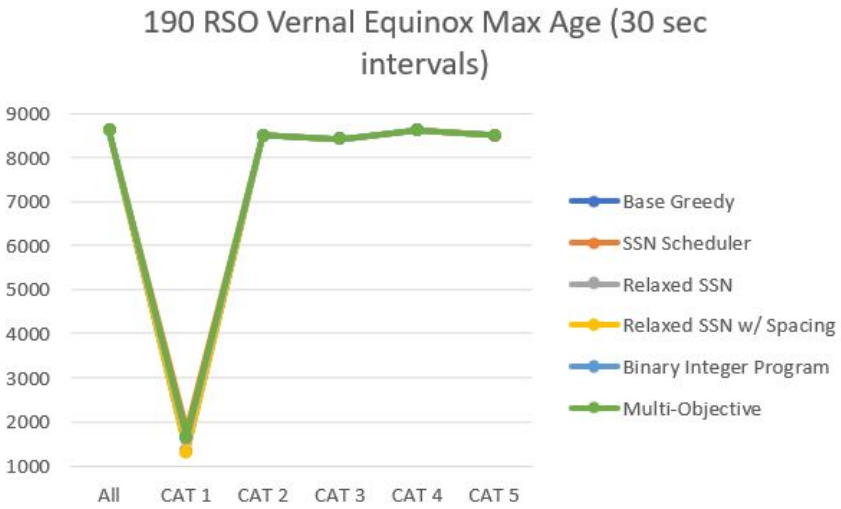

Figure 38. 190 RSO Vernal Equinox Max Age

4.7.2.1 Base Model Greedy

Similar to the large population, there is little difference in comparison to the other models, however, in this scenario the greedy model performs level with the rest because RSO's in the population have sufficient SNR.

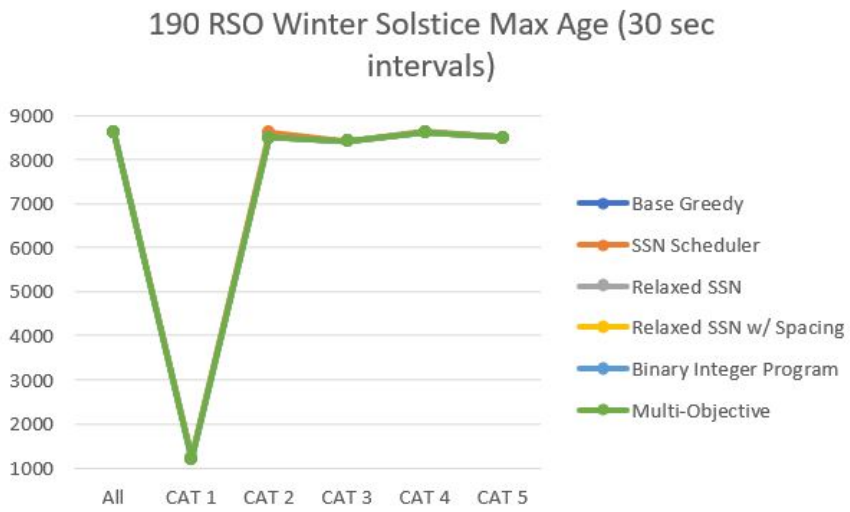

Figure 39. 190 RSO Winter Solstice Max Age

4.7.2.2 SSN Scheduler Model

The SSN Scheduler performs fairly on par with all other schedulers the on all three days. On Summer Solstice, CAT 5 max age is slightly worse than the rest.

4.7.2.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model performs about the same as all other models. It performs slightly better in CAT 1's on Summer Solstice, but performs slightly worse in CAT 5's along with the SSN Scheduler.

4.7.2.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model performs about the same as all other models, but is slightly better on Summer Solstice and Vernal Equinox.

4.7.2.5 Binary Integer Program Model

The Binary Integer Program Model performs about the same as all other schedulers. It outperforms the second relaxed model slightly in CAT 1's on Winter Solstice,

but otherwise is behind on other dates.

4.7.2.6 Multi-Objective Binary Integer Program Model

The multi-objective model's results are slightly worse than the Binary Integer Program, but only by a few intervals.

4.7.2.7 Conclusion

Max age does not change drastically with any of the models, though up to a 27% decrease of max age of CAT 1's, up to 1% improvement in CAT 2's, and up to 4% improvement in CAT 5's is seen in novel schedulers. They all reduce the max age of CAT 1's and the Relaxed SSN Scheduler with Spacing model slightly outperforms the rest.

4.8 Run Times

Run times measure the amount of time the scheduler script takes to create a viable solution. It compares how fast a scheduler is one to another.

4.8.1 3968 RSO's

Each schedulers results are detailed in the following subsections. Overarching trends are that the Base Model Greedy scheduler provides the fastest solution, which is expected with the least complex code. The Relaxed SSN Scheduler with Spacing Model runs the slowest, but only by a few seconds in the large population.

4.8.1.1 Base Model Greedy

The Base Model Greedy Scheduler consistently run the fastest. On both solstices it ran over 100 seconds faster than the closest competitor, though on the Vernal

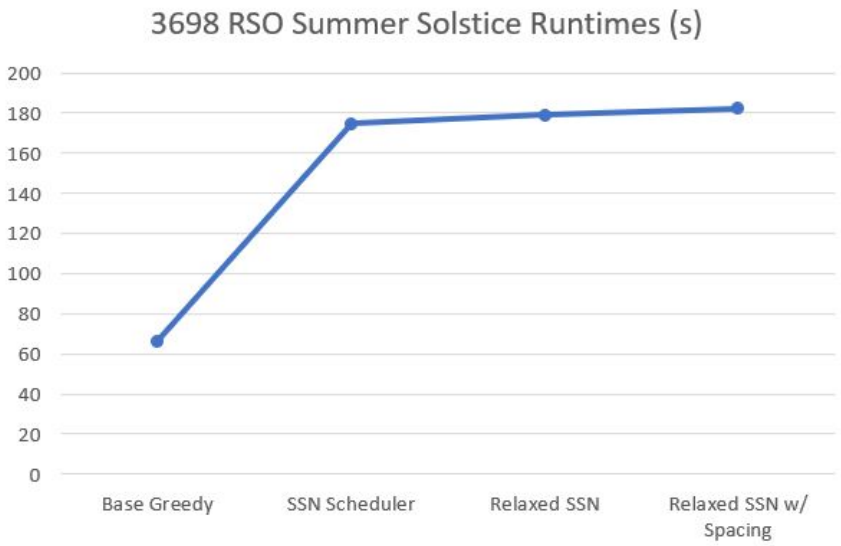

Figure 40. 3968 RSO Summer Solstice Run Times

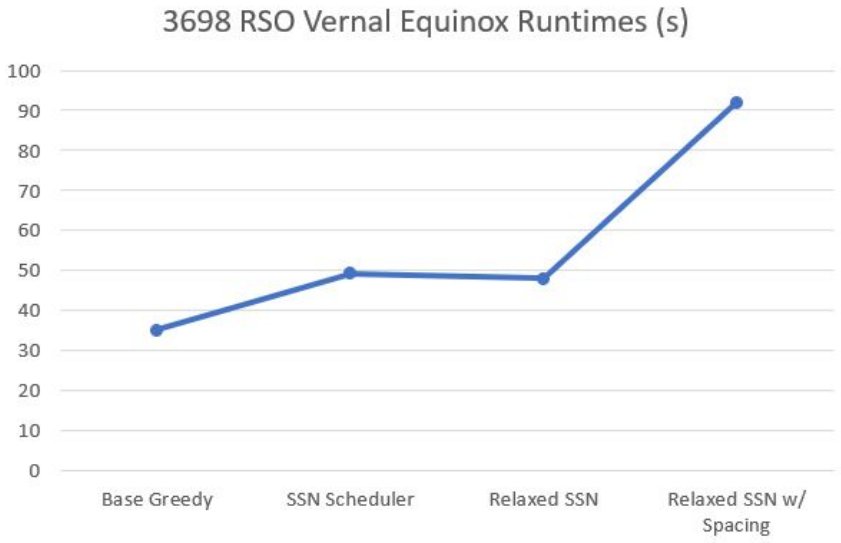

Figure 41. 3968 RSO Vernal Equinox Run Times

equinox it was only 15 seconds faster. This result is as expected as the base model has the least complex code.

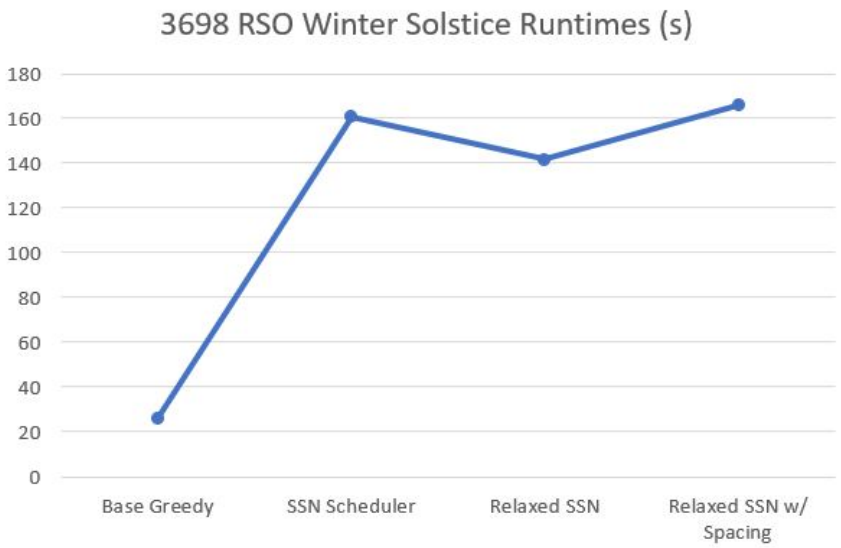

Figure 42. 3968 RSO Winter Solstice Run Times

4.8.1.2 SSN Scheduler Model

The SSN Scheduler runs generally in the same amount of processing time as the relaxed models. This scheduler typically runs faster than the Relaxed SSN Scheduler Model with Spacing. It is expected to run slower than the Relaxed SSN Scheduler Model because it has more complex code due to the additional constraints, however, on the Summer Solstice it ran faster in simulation tests.

4.8.1.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model runs slower than the SSN Scheduler model on Summer Solstice, but slightly runs faster on the other two dates. In general, this script is expected to run faster than the SSN Scheduler Model as its logic and code are less complex than the SSN Scheduler Model.

4.8.1.4 Relaxed SSN Scheduler Model with Spacing

The second relaxed model is the slowest running scheduler in the large population scenario. However, the run time is only a few seconds behind the rest of the schedulers.

4.8.1.5 Conclusion

In the large RSO population, the Base Model Greedy provides the fastest solution, as expected due to having the least complexity. The next fastest is the Relaxed SSN Scheduler Model, followed by the SSN Scheduler and then the Relaxed SSN Scheduler with Spacing Model, which follows closely behind the others.

4.8.2 190 RSO's

Detailed analysis of each scheduler for the 190 RSO scenarios is included in the following subsections. Overarching trends are that both the Binary Integer Program Model and the Multi-objective Binary Integer Program Model take significantly longer to execute than the other models. Both scripts take approximately 10-15 times as long.

4.8.2.1 Base Model Greedy

The Base Model Greedy script runs the fastest of all schedulers. This is expected as it has the least complex code.

4.8.2.2 SSN Scheduler Model

The SSN Scheduler runs similarly to the two relaxed models, each having run faster on different simulation days, but slower than the greedy scheduler.

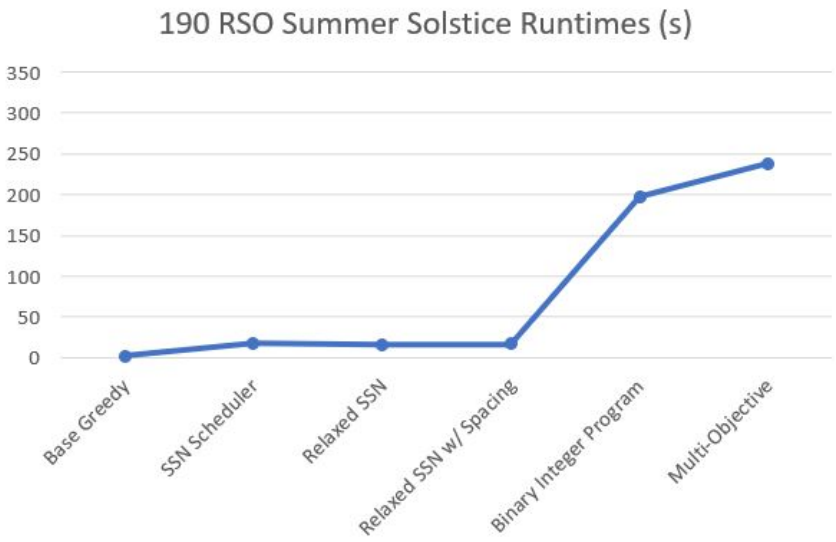

Figure 43. 190 RSO Summer Solstice Run Times

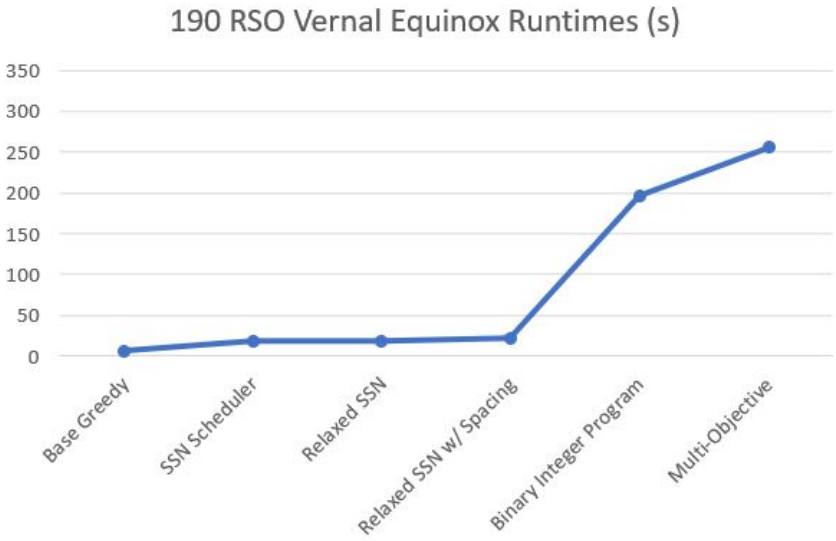

Figure 44. 190 RSO Vernal Equinox Run Times

4.8.2.3 Relaxed SSN Scheduler Model

The first relaxed model scheduler runs only a few seconds behind the SSN Scheduler on two of the simulation dates and a second faster on the Summer Solstice.

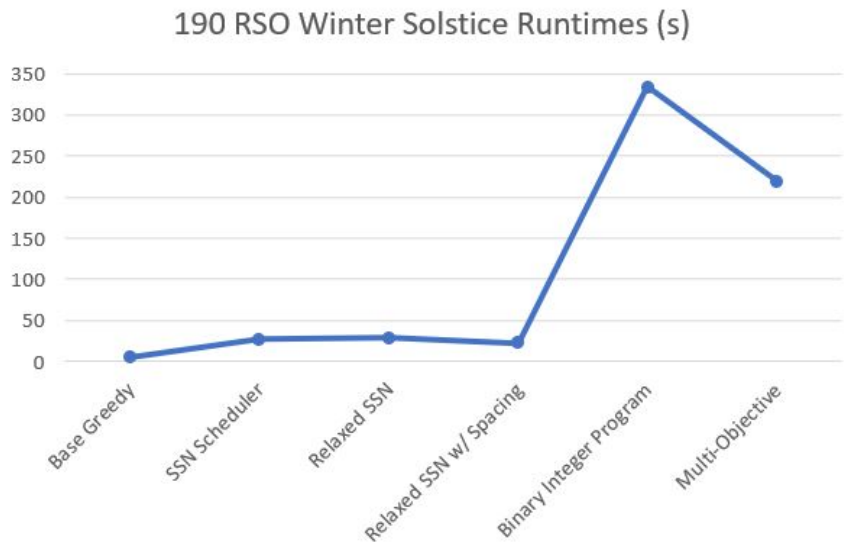

Figure 45. 190 RSO Winter Solstice Run Times

4.8.2.4 Relaxed SSN Scheduler with Spacing Model

The relaxed model with spacing runs a few seconds behind the first typically, with exception on the Winter Solstice, where it ran slightly faster than both the relaxed model and SSN Scheduler Model.

4.8.2.5 Binary Integer Program Model

The Binary Integer Program runs significantly longer than the heuristic style schedulers. On each day, the Binary Integer Program Model runs approximately 10-15 times as long to create a solution.

4.8.2.6 Multi-Objective Binary Integer Program Model

The Multi-Objective Binary Integer Program Model also runs significantly longer than the heuristic style schedulers and runs a few seconds slower than the Binary Integer Program Model.

4.8.2.7 Conclusion

The Base Model Greedy scheduler runs faster than all others. The SSN Scheduler and both relaxed models follow close behind, especially when compared to that of the integer programs, but with this number of RSO's the difference is difficult to tell which is truly fastest. Additional runs to average the run times would be required to determine which script runs the fastest at this number of RSO's. The Relaxed SSN Scheduler Model is expected to run the fastest of the three, followed by the SSN Scheduler, then the Relaxed SSN Scheduler with Spacing Model due to code complexity. The integer programs take 10-15 times as long as the other schedulers to develop a solution. This result is consistent with expectations. Heuristic schedulers are, by definition, intended to run quickly.

4.9 Multi-Objective Weights

A decision maker may choose to apply different weighting to each objective in the multi-objective model. In the previous runs, the multi-objective model was run with 50% weighting for each objective. The model was additionally run on the Winter Solstice with the weights shown in Table 7, where Weight 1 represented the objective to schedule RSO's with higher priorities and penalize over or under observing and Weight 2 represents the objective of observing RSO's with higher initial time between observations.

Table 7. Multi-Objective Binary Integer Program Weight Combinations

Run	Weight 1	Weight 2
1	99%	1%
2	75%	25%
3	50%	50%
4	25%	75%
5	1%	99%

Running at each of these weight settings produced results with the same number of observations in each category area and same number of RSO's meeting thresholds in each category. Additionally, the max ages remained the same with each weight combination. Different weights did produce changes to mean age, the results of which are shown in Table 8.

Table 8. Multi-Objective Binary Integer Program Mean Ages with Different Weight Combinations

	1%/99%	25%/75%	50%/50%	75%/25%	99%/1%
All:	3838	3832	3846	3850	3847
CAT 1:	1202	1202	1202	1202	1202
CAT 2:	3429	3426	3466	3482	3463
CAT 3:	3370	3355	3367	3419	3351
CAT 4:	3587	3554	3582	3562	3582
CAT 5:	4207	4213	4213	4219	4218

While the objective to observe RSO's with higher initial time between observations aimed to reduce mean age and did so, the idea weighting combination was 25% for the first objective of scheduling based on priority and thresholds and 75% for the objective to observe higher initial time between observations. This combination provided the lowest mean age for most categories, the only exception being CAT 5, which could be lowered slightly by leaning further towards objective 2 at expense of other categories' mean age.

4.10 Summary

The results show the SSN Scheduler under performs compared to the other models in several areas. In terms of observations across various categories it has the smallest diversity, choosing to observe only CAT 1 or CAT 2 RSO's. It also provides the worst capability meeting RSO thresholds, or reducing mean time between observations. The Relaxed SSN Scheduler with Spacing Model performs the best in several areas

including maximizing total RSO's meeting observation thresholds, reducing mean and max age, while having a decently short execution time. The integer programs provide results that come close or perform better in some areas, however this is offset by a considerably long run time. Novel schedulers improve overall threshold numbers by 3%-20% and decrease mean age up to 12%-33% overall. The Multi-Objective Binary Integer Program Scheduler Model shows that varying the weighting for each objective can lead towards reducing the mean age, but has little effect on other metrics. Tables with result data are in Appendix B. Chapter V provides methodology and analysis of multi-sensor scenarios. Chapter VI covers conclusions drawn from this research.

V. Multi-sensor Methodology and Analysis

5.1 Overview

This chapter discusses the problem formulation, model development, and implementation of each model in a multi-sensor domain. The chapter explains the methods adjusted from the single sensor scheduler designs. The areas of measurement remain the same as Chapter III and include mean and max age between observations, total number and sum of RSO's observed by each priority category, and total number of RSO's which met target threshold values in one day. The chapter concludes with analysis of the experimental trials conducted.

5.2 Methodology

5.2.1 Approach Overview

Chapter III covered application of all schedulers on a single sensor. The previous theses conducted by Stern and Wachtel [9], Felten [10], Bateman [11], and Basraoui [48], assumed sensors developed a schedule centrally rather than separately. The 2004 SD 505-1 indicates RSO prioritization is centrally developed and then distributed to each sensor as a tasking list, but sensors individually develop schedules. To compare 2004 SD 505-1 scheduler against novel schedulers on a multi-sensor scale, each of the novel schedulers' were modified. Bateman's code [11], from which Relaxed SSN Scheduler model and the Relaxed SSN Scheduler Model with Added Spacing were developed, already included multi-sensor capability, in that it included the capability at each time interval to schedule an RSO observation by each sensor. This capability was carried over into the Relaxed SSN Scheduler model and the Relaxed SSN Scheduler Model With Added Spacing, so that more than one sensor could be considered in a centralized scheduler. To simulate the decentralized scheduling, the SSN Scheduler

model was run at each sensor separately based on the three different locations and the results of the three separate schedules developed were combined. Both the Binary Integer Program Model and the Multi-Objective Binary Integer Program Model were adapted to include multiple sensors within the constants, decision variables, and objective functions.

5.2.2 Ground Sensor Generation

In addition to the sensor created at Socorro, NM (Latitude 33.82, Longitude -106.66, Altitude 1403m), two additional sensors were added at Diego Garcia, British Indian Ocean Territory (Latitude -7.3195, Longitude 72.4229, Altitude 0m) and Maui, HI (Latitude 20.7083, Longitude -156.2571, Altitude 3052m) with ground sensor characteristics mimicking that of the GEODSS sensors to complete their network [26]. Once again, they were created within STK with solar exclusion angle of 40, lunar exclusion angle of 10, minimum elevation angle of 20 and constrained to only operate in umbra, and 3 telescopes at each location. New custom vectors and angles were created for every RSO/sensor platform pair, the same reports as in the single sensor experiments were created for each of the three experiment days.

5.2.3 Scenario Limitations

As mentioned in Chapter IV, the hardware utilized to run both Binary Integer Models was a 32 bit system and limited in the RAM available to the Python software. Modeling three sensors within the Binary Integer models required three times as many equations, therefore, the population size of the RSO's was reduced to 50, a size which the hardware could handle running the Multi-Objective Binary Integer Model Scheduler. To determine the ideal RSO's for the experiment, the full 3,968 population was simulated individually on each day at each sensor location. A list

of visible RSO's meeting the SNR threshold were recorded for each location on each day and the three lists were summed together and sorted by number of stations the RSO was visible from. In this way, the top 50 RSO's that were visible at either two or three of the ground sensors were determined and used for the multi-sensor experiments. The breakdown of priorities for the population can be seen in Table 9, and each was assigned a random initial time between observations.

Table 9. 50 RSO Category Breakdown

Category/Suffix	50 RSO
1A	0
1B	1
1C	1
1D	0
1E	0
2A	2
2B	2
2C	1
2D	1
2E	0
3A	0
3B	1
3C	2
3D	0
3E	0
4A	8
4B	3
4C	1
4D	4
4E	2
5A	2
5B	3
5C	7
5D	6
5E	3
1s	2
2s	6
3s	3
4s	18
5s	21

5.2.4 Base Model Greedy Scheduler

The Base Model Greedy Scheduler already included capability to run at multiple sensors and loops additionally during each interval for each sensor. This means the greedy scheduler develops a centralized schedule. At each time interval, the scheduler

searches a visible RSO with the largest time between observations at the first sensor. Then, the scheduler does the same at the next sensor and iterate until it has selected RSO's for each sensor. The scheduler then increments the time counter and repeats the process on the next time interval until the end of the scenario. The adjusted flow chart can be seen in Figure 46. Again, note that the Base Greedy Model, which captures prior research methodologies, does not include an SNR filter and thus has greater flexibility in available RSO's than other scheduler models.

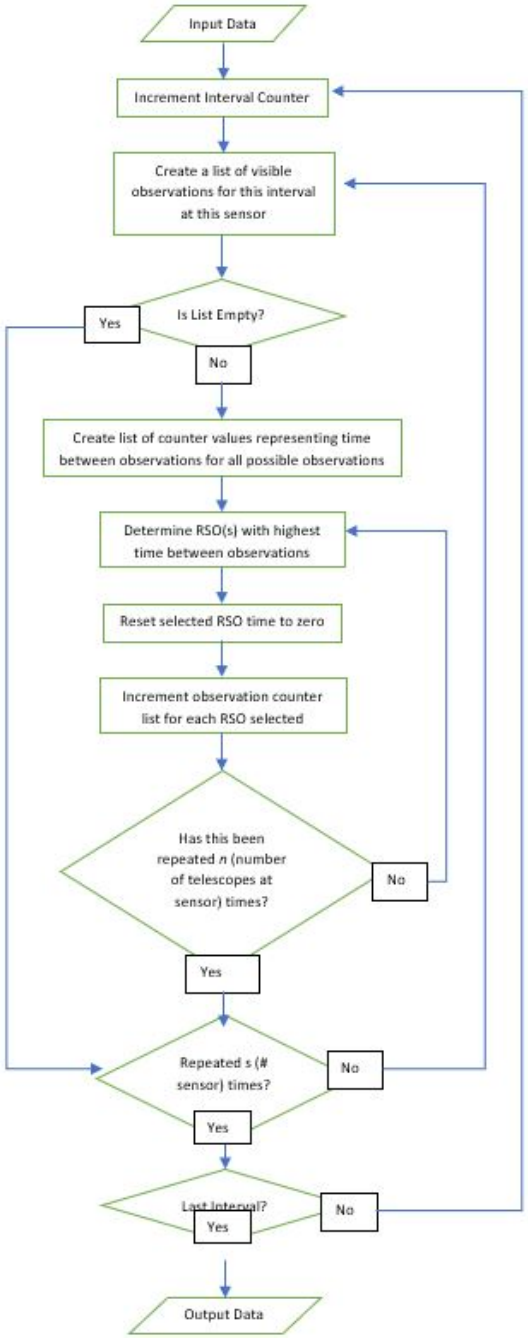

Figure 46. Multi-Sensor Base Model Greedy Scheduler Flow Chart

5.2.5 SSN Scheduler Model

The SSN Scheduler Model does not change from the single sensor design because it represents the conditions described in 2004 SD 505-1, where scheduling is decen-

tralized and each sensor creates its own schedule. It is run separately on each sensor, and the resulting observations for each sensor are combined. The time between observations for each RSO from each sensor run is compared and the lowest time between observation was chosen to represent the most recent observation of the RSO by any sensor. Since no centralized deconfliction is occurring, RSO's can be over observed by the schedules developed.

5.2.6 Relaxed SSN Model Scheduler

The Relaxed SSN Model Scheduler simulates a centralized scheduler design. The Relaxed SSN Model Scheduler iterates between sensors at each interval. Since this is a centralized scheduler, the Relaxed SSN Model Scheduler selects a different RSO for observation at each of the multiple sensors during the same interval unless all RSO's have met thresholds and the RSO in question has the highest priority category. Figure 47 depicts the flow chart.

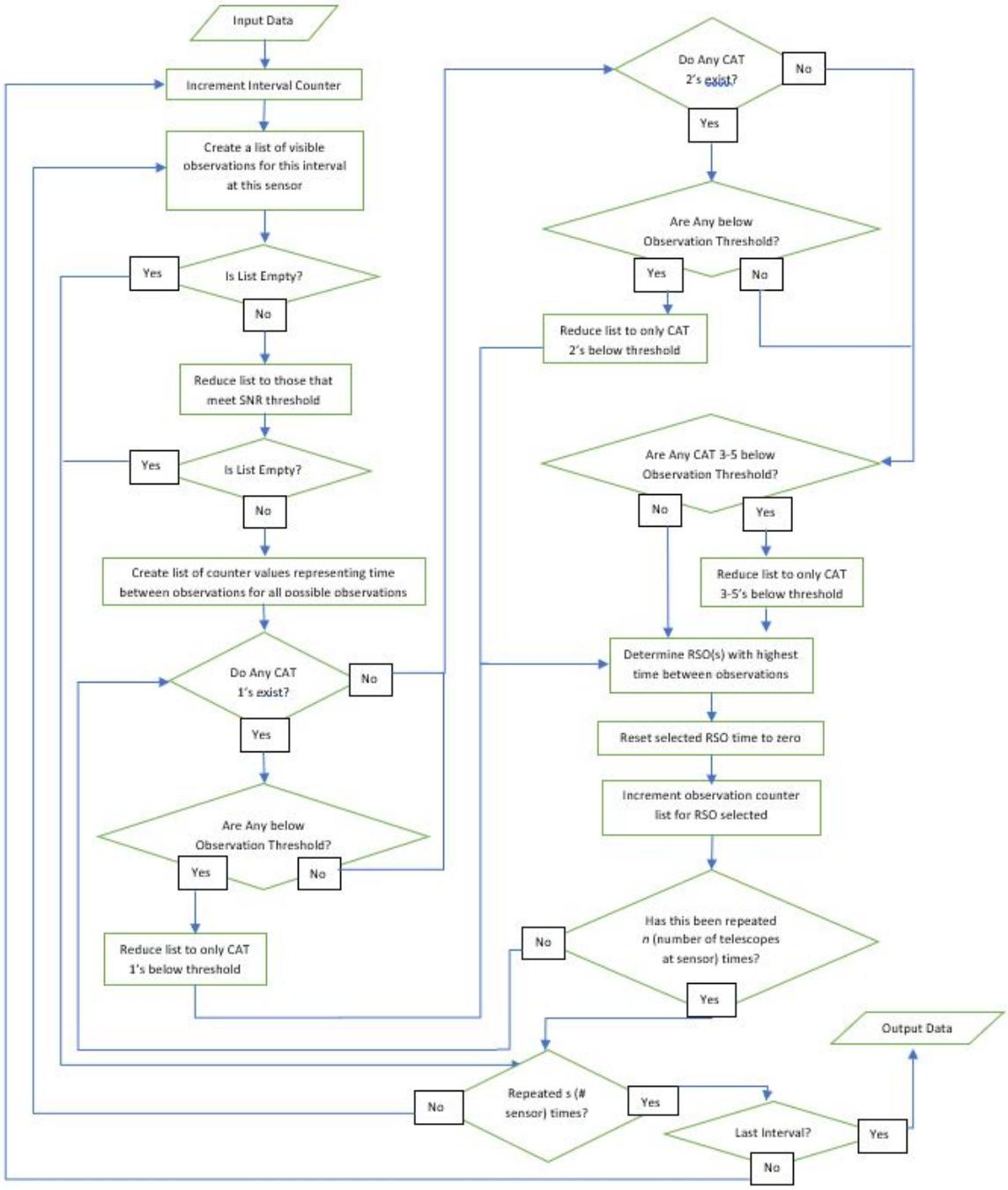

Figure 47. Multi-Sensor Relaxed SSN Model Scheduler Flow Chart

5.2.7 Relaxed SSN Scheduler Model with Added Spacing

This scheduler model operates the same as the relaxed model in 5.2.6 with added constraint of attempting to enforce separation between observation times, see Figure

48 for this scheduler's flowchart. Because the scheduler is centralized, sensors first look for an RSO that has not been observed in the last 60 minutes. If all visible RSO's have been viewed in the last 60 minutes, the scheduler selects the highest priority RSO that has not yet met threshold.

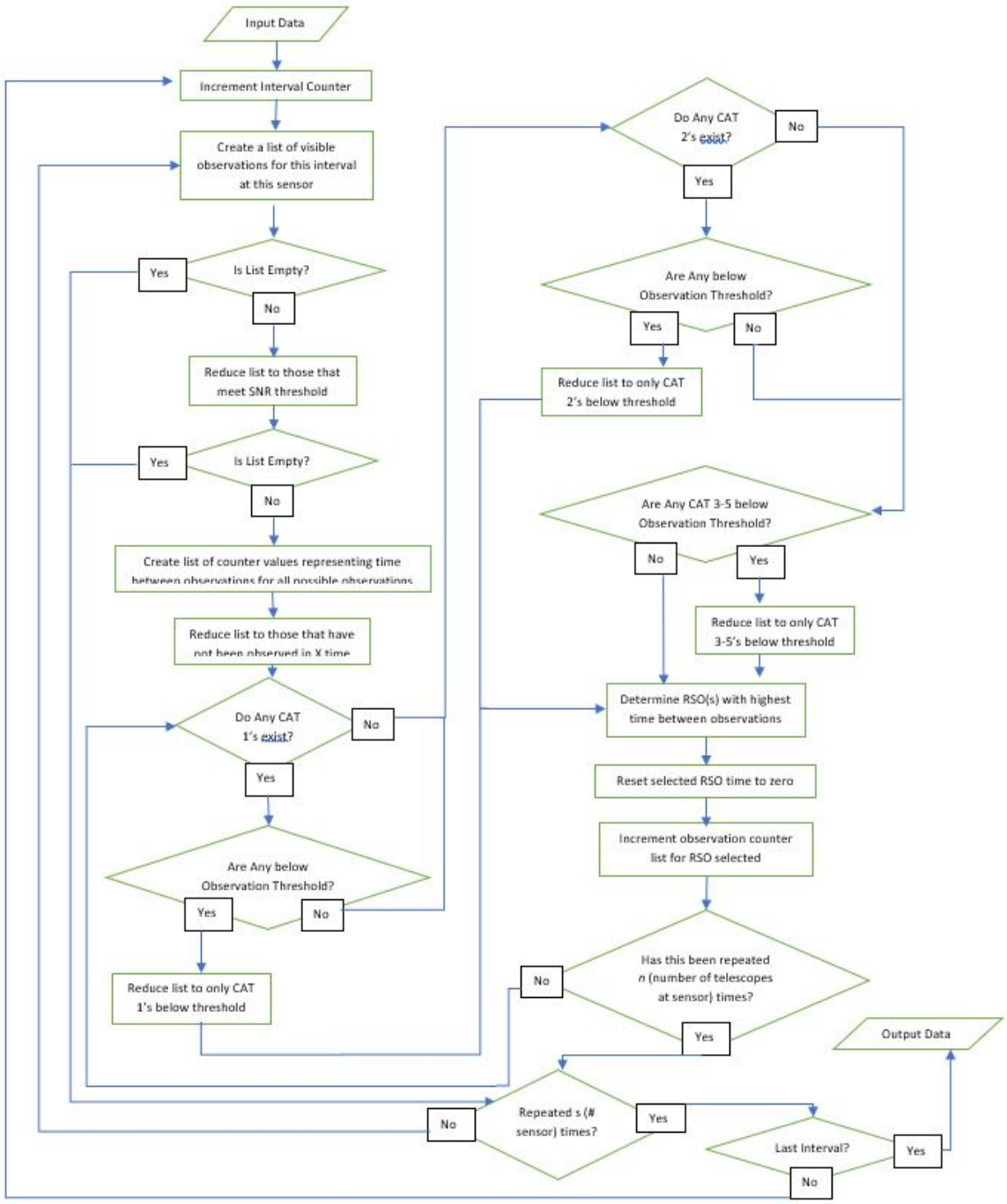

Figure 48. Multi-Sensor Relaxed SSN Scheduler Model with Added Spacing Flow Chart

5.2.8 Binary Integer Model Scheduler

The Binary Integer Model also includes a centralized scheduler. In order to modify the single sensor model, an additional dimension is added for sensors. Constraint Equation 35 is included for all sensors within the set because each sensor can observe three RSO's at each interval. Constraint Equation 36 includes an additional summation because each sensor's observation counts towards meeting threshold requirements.

Constants:

c_i: penalty for observing object i over threshold number in one day

d_i: penalty for observing object i under threshold number in one day

g_{sij}: binary element sij in visible matrix, $g_{sij} = 1$ if visible, 0 otherwise

h_i: value of observing object i

t_i: observation threshold value of object i

Decision Variables:

x_{sij}: # of observations sensor s takes of object i at time interval j in one day

y_i^+: # of observations above threshold sensor takes of object i

y_i^-: # of observations under threshold sensor takes of object i

Where:

$I := \{i | i = 1, 2, ..., 3967, 3968\}$ (The set of all RSO's)

$J := \{j | i = 1, 2, ..., 2779, 2880\}$ (The set of 30 second time intervals)

$S := \{s | s = 1, 2, or 3\}$ (The set of ground sensors)

Objective Function:

$$\text{Maximize:} z = \sum_{i=1}^{3968}((\sum_{j=1}^{2880}\sum_{s=1}^{3} h_i x_{sij}) - c_i y_i^+ - d_i y_i^-) \tag{34}$$

Constraints:

$$\sum_{i=1}^{3968} x_{sij} \leq 3, \forall j \in J, \forall s \in S \tag{35}$$

$$(\sum_{j=1}^{2880} \sum_{s=1}^{3} x_{sij}) - y_i^+ + y_i^- = t_i, \forall i \in I \tag{36}$$

$$x_{sij} \leq g_{sij}, \forall i \in I, \forall j \in J, \forall s \in S \tag{37}$$

$$y_i^+, y_i^- \geq 0, \forall i \in I \tag{38}$$

$$x_{sij} = 0 \text{ or } 1, \forall i \in I, \forall j \in J, \forall s \in S. \tag{39}$$

5.2.9 Multi-Objective Model Scheduler

The Multi-Objective Model Scheduler is also a centralized scheduler, therefore a dimension was also added to the model. Constraint Equation 44 includes a summation because the second objective can be fulfilled by any sensor.

Constants:

c_i: penalty for observing object i over threshold number in one day

d_i: penalty for observing object i under threshold number in one day

g_{sij}: binary element sij in visible matrix, $g_{sij} = 1$ if visible, 0 otherwise

h_i: value of observing object i

t_i: observation threshold value of object i

m_i: value of observing object i according to its time between observations

k: normalization constant for second objective

Decision Variables:

x_{sij}: # of observations sensor s takes of object i at time interval j in one day

y_i^+: # of observations above threshold any sensor takes of object i

y_i^-: # of observations under threshold any sensor takes of object i

q_i: whether or not object i has been chosen for at least one observation

Where:

$I := \{i | i = 1, 2, ..., 3967, 3968\}$ (The set of all RSO's)

$J := \{j | i = 1, 2, ..., 2779, 2880\}$ (The set of 30 second time intervals)

$S := \{s | s = 1, 2, or 3\}$ (The set of ground sensors)

Objective Function:

$$\text{Maximize:} z = w_1(\sum_{i=1}^{3968}((\sum_{j=1}^{2880}\sum_{s=1}^{3} h_i x_{sij}) - c_i y_i^+ - d_i y_i^-)) + w_2(\sum_{i=1}^{3968} km_i q_i) \tag{40}$$

Constraints:

$$\sum_{i=3}^{3968} x_{sij} \leq 3, \forall j \in J, \forall s \in S \tag{41}$$

$$(\sum_{j=1}^{2880}\sum_{s=1}^{3} x_{sij}) - y_i^+ + y_i^- = t_i, \forall i \in I \tag{42}$$

$$x_{sij} \leq g_{sij}, \forall i \in I, \forall j \in J, \forall s \in S \tag{43}$$

$$q_i \leq \sum_{j=1}^{2880}\sum_{s=1}^{3} x_{sij}, \forall i \in I \tag{44}$$

$$y_i^+, y_i^- \geq 0, \forall i \in I \tag{45}$$

$$x_{sij} = 0 \text{ or } 1, \forall i \in I, \forall j \in J, \forall s \in S \tag{46}$$

$$q_i = 0 \text{ or } 1, \forall i \in I. \tag{47}$$

5.2.10 Multi-Objective Normalization Factor

Each of the Multi-Objective Binary Integer Program Model runs were made initially with a k value of 1. The values of each objective were totaled and used to determine the appropriate normalization factor, shown in Table 10. Additional runs were made on the Winter Solstice to determine if adjusting the weights of each objective value would change the k value calculated, however, the k values remained

constant.

Table 10. Multi-Sensor Multi-Objective Binary Integer Program Model K Values

Date/Weights	Single Sensor K Value
Summer Solstice ($w1 = 0.5, w2 = 0.5$)	380.86
Vernal Equinox ($w1 = 0.5, w2 = 0.5$)	391.43
Winter Solstice ($w1 = 0.5, w2 = 0.5$)	521.34

5.3 Analysis

Relaxed versions of the SSN Scheduler are shown to improve mean age, increased number of RSO's meeting observation thresholds, and allowing the schedulers to view RSO's of all categories. The SSN Scheduler, on the other hand performs poorly in a multi-sensor situation as the other schedulers have the advantage of centralized scheduling to reduce overlap.

5.3.1 Total Observed

In a small population with three sensors, the SSN Scheduler and relaxed models behave similarly to trends seen in section 4.4. The integer programs conversely choose to observe a large number of CAT 2 and 4 RSO's.

5.3.1.1 Base Model Greedy

Like in the 3968 RSO scenarios seen in section 4.4.1.1, this model generally made observations in proportion to the population. CAT 5 RSO observations were the highest category viewed.

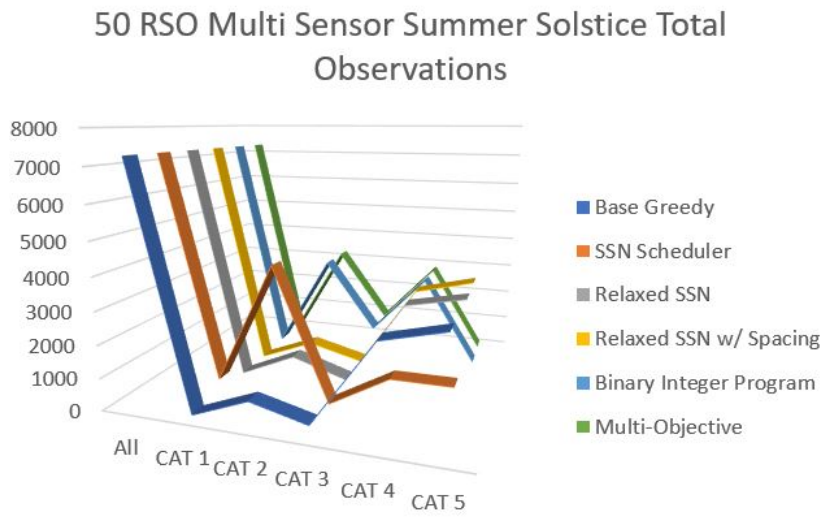

Figure 49. 50 RSO Summer Solstice Total Observations

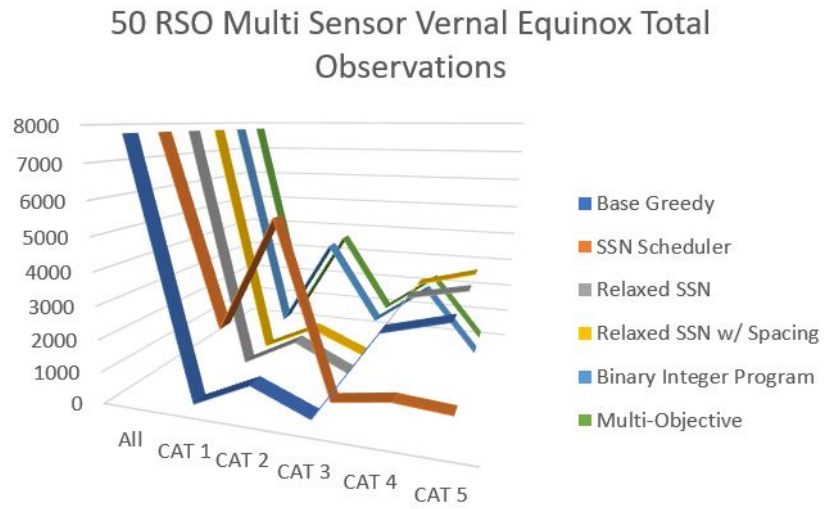

Figure 50. 50 RSO Vernal Equinox Total Observations

5.3.1.2 SSN Scheduler Model

The SSN Scheduler Model focused primarily on CAT 2's during the Summer Solstice, a mix with the larger portion being CAT 1's during the Winter Solstice, and then a mix with majority being Cat 2's during the Vernal Equinox. The focus on CAT 2's is likely due to low SNR ratios at various sensors as well as the small

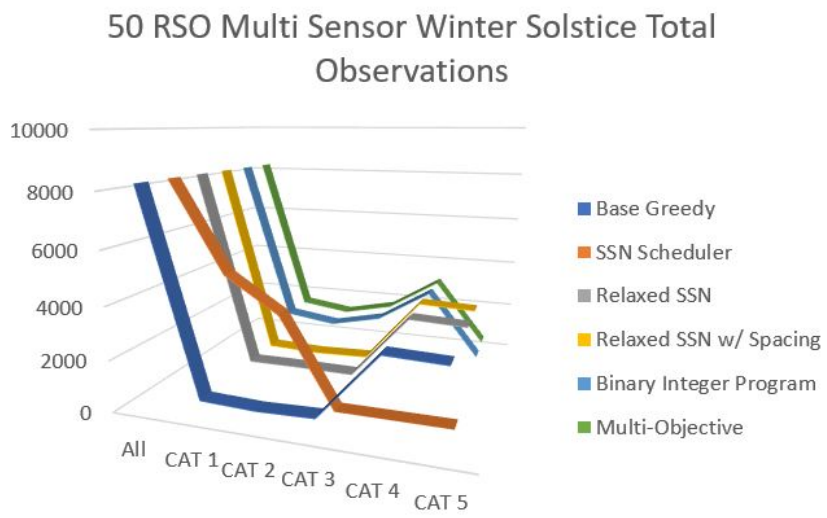

Figure 51. 50 RSO Winter Solstice Total Observations

population of CAT 1's.

5.3.1.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model viewed significantly less CAT 2's and instead spread the observations to CAT 4 and Cat 5 RSO's.

5.3.1.4 Relaxed SSN Scheduler with Spacing Model

The second relaxed model performed very similarly to the first Relaxed SSN Scheduler Model with very little variation.

5.3.1.5 Binary Integer Program Model

The Binary Integer program viewed significantly less CAT 5's instead increasing the CAT 2's and CAT 3's due to their higher worth value. A larger increase in CAT 1 and CAT 3's is seen in Winter Solstice at the expense of CAT 2's.

5.3.1.6 Multi-Objective Binary Integer Program Model

The Multi-Objective Model with equally weighed objectives performed very similarly to the Binary Integer Program Model.

5.3.1.7 Conclusion

Similar to single sensor runs in section 4.4, the SSN Scheduler focused on CAT 1 and 2 RSO's while the relaxed models spread the observations across the categories, viewing more lower priority RSO's. The integer programs uniquely chose to view a large amount of CAT 2 and 4's, with very few CAT 5 observations. These results further indicate how the SSN Scheduler poorly observes lower priority targets and over observes higher priority targets. Additionally, the integer programs performance to view a large number of CAT 2 and 4's indicate several CAT 2's were close to meeting thresholds and were able to met through multi-sensor centralized scheduling, providing a better value than meeting fewer CAT 1's. At the same time, this knowledge is significant in that if the overarching goal is to retain CAT 1 observation performance, the integer models may not be the best scheduler selection, however, in section 5.3.2, results show that thresholds met are the same or better.

5.3.2 Total RSO's Meeting Observation Threshold

Both integer program schedulers perform the best in fulfilling maximum number of thresholds across categories. At least 10% increase in overall threshold numbers is seen in novel schedulers with up to a 66% improvement in CAT 3-5's. The SSN Scheduler performs consistently the worst and demonstrates exceptionally lower results during Winter Solstice.

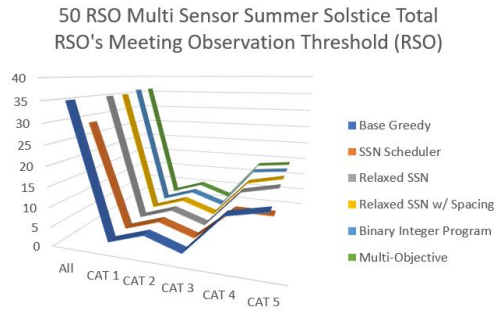

Figure 52. 50 RSO Summer Solstice Total RSO's Meeting Observation Threshold

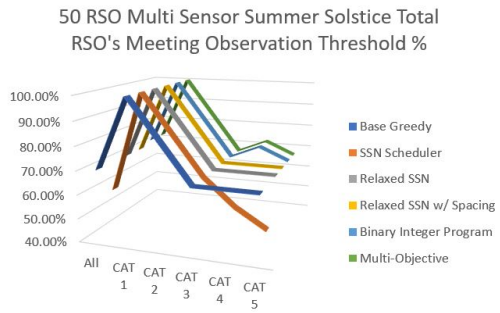

Figure 53. 50 RSO Summer Solstice Total RSO's Meeting Observation Threshold %

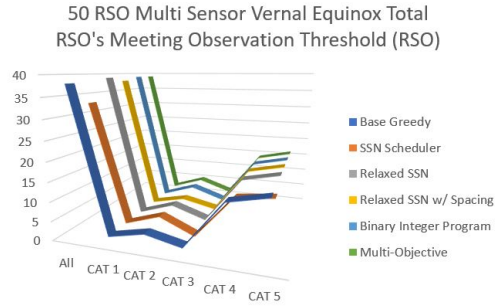

Figure 54. 50 RSO Vernal Equinox Total RSO's Meeting Observation Threshold

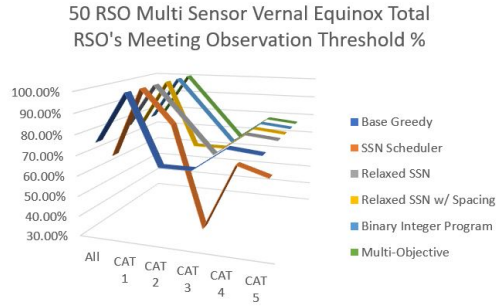

Figure 55. 50 RSO Vernal Equinox Total RSO's Meeting Observation Threshold %

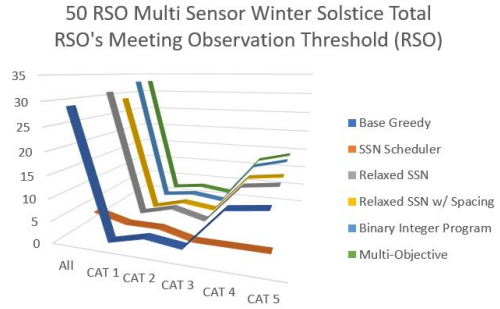

Figure 56. 50 RSO Winter Solstice Total RSO's Meeting Observation Threshold

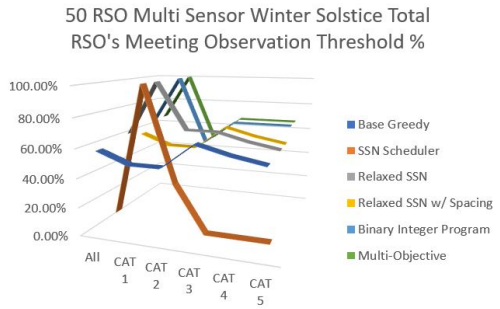

Figure 57. 50 RSO Winter Solstice Total RSO's Meeting Observation Threshold %

5.3.2.1 Base Model Greedy

In this population, the base model performs with almost the same results as all other schedulers except the SSN Scheduler. A balance of RSO's proportional to the

number in each category meet threshold.

5.3.2.2 SSN Scheduler Model

The SSN Scheduler under performs in CAT 4 and 5's compared to the other schedulers, bringing the total number meeting threshold down. During Winter Solstice, the SSN Scheduler has an extremely low number meeting threshold, nearly one seventh that of the other scenario dates.

5.3.2.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler Model receives very close values to those seen in the greedy model.

5.3.2.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model is one of the only models unable to meet 100% threshold for CAT 1's. This is a negative indicator showing the SSN Scheduler Model, which is intended to provide highest focus on CAT 1's, is unable to perform well in CAT 1's in a multi-sensor setting. The decentralized schedule negatively affects the scheduler's performance because each sensor views the same RSO's, causing unnecessary redundancy.

5.3.2.5 Binary Integer Program Model

The Binary Integer Program Scheduler performs slightly better than the four previous models, fulfilling an increased amount of CAT 4's to meet threshold. This performance increase is more pronounced at Winter Solstice.

5.3.2.6 Multi-Objective Binary Integer Program Model

Once again, the multi-objective model performs almost identically to the Binary Integer scheduler.

5.3.2.7 Conclusion

In terms of enabling RSO's to meet observation thresholds, the integer program schedules perform the best, enabling 100% of CAT 1's to meet threshold while maintaining a higher level meeting threshold in other categories. At least 10% increase in overall threshold numbers is seen in novel schedulers with up to a 66% improvement in CAT 3-5's. The SSN Scheduler performs the worst in meeting thresholds, exceptionally poorly during Winter Solstice.

5.3.3 Mean Age

The SSN Scheduler is shown to not only perform the worst overall, but also performs significantly worse for CAT 1 and 2, the high priority categories it focuses. This is probably resulting from the fact that this model is individually scheduling each sensor instead of leveraging a centralized scheduler. Mean age of all RSO's improves at least 51% and at least 46% in each category using novel schedulers. Both relaxed models along with the base model provide the best mean age reduction.

5.3.3.1 Base Model Greedy

The Base Model Greedy performs on par with most other models except the SSN Scheduler.

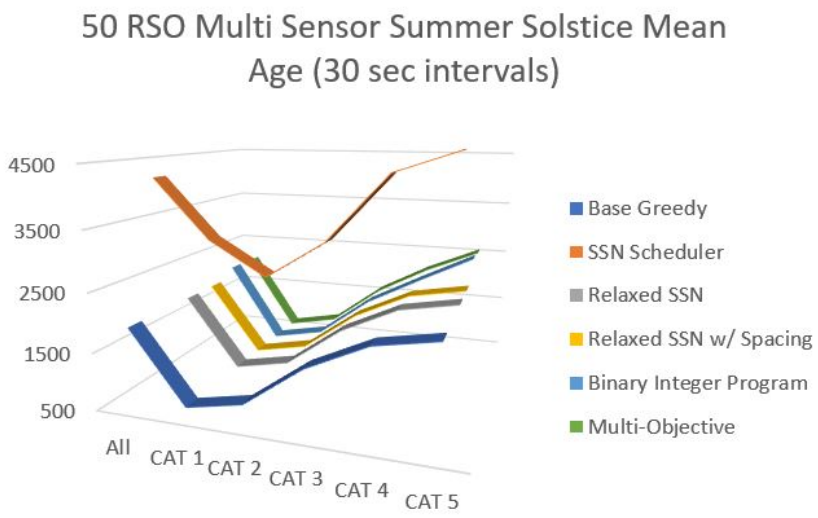

Figure 58. 50 RSO Summer Solstice Mean Age

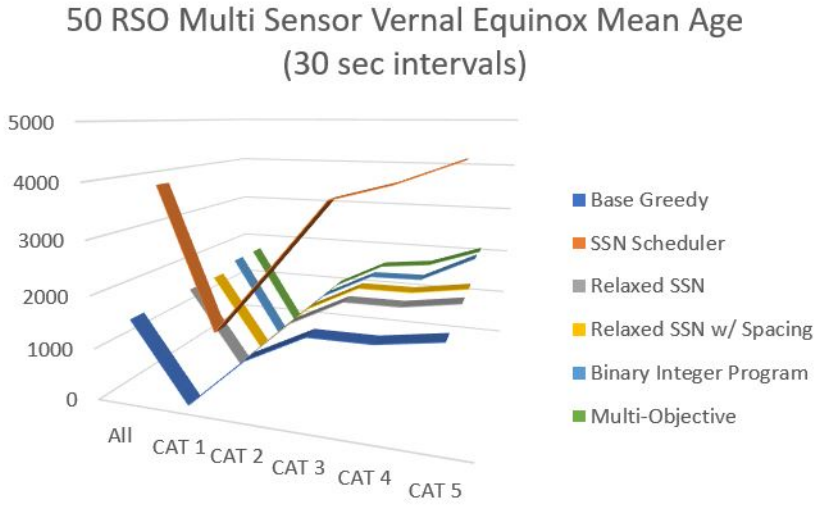

Figure 59. 50 RSO Vernal Equinox Mean Age

5.3.3.2 SSN Scheduler Model

The SSN Scheduler performs the worst at reducing mean age. Each category is at least 1000 to 2000 intervals higher than other schedulers.

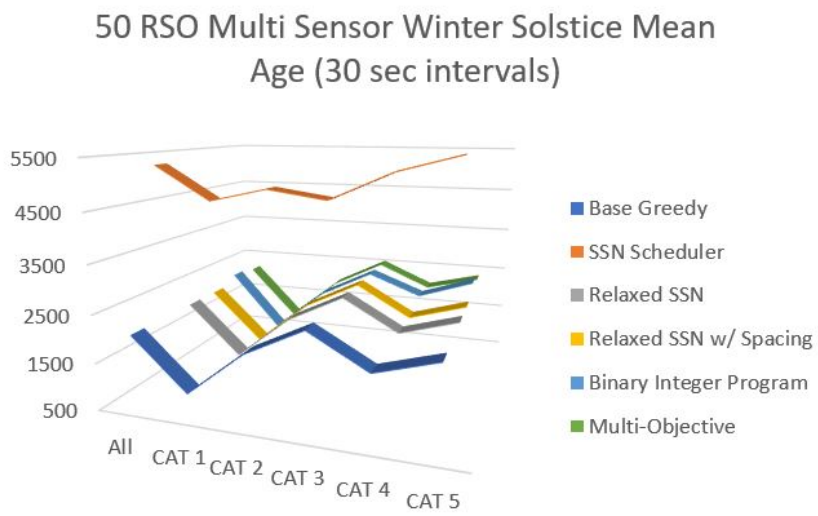

Figure 60. 50 RSO Winter Solstice Mean Age

5.3.3.3 Relaxed SSN Scheduler Model

The Relaxed SSN Scheduler models performs on par with the Base Model Greedy scheduler. These results are the best within the schedulers in reducing mean time.

5.3.3.4 Relaxed SSN Scheduler with Spacing Model

The Relaxed SSN Scheduler with Spacing Model also performs on par with the base model, meaning it performs better than most of the other models.

5.3.3.5 Binary Integer Program Model

The Binary Integer Program Model performs worse than the relaxed models, having a higher mean age in CAT 5's.

5.3.3.6 Multi-Objective Binary Integer Program Model

The multi-objective model is similar to the Binary Integer Program Model, but performs slightly better on each scenario day, nearly reaching the same values as the

relaxed models during Winter Solstice.

5.3.3.7 Conclusion

Both relaxed models using a centralized scheduler along with the base greedy model perform the best at reducing mean age. Mean age of all RSO's improves at least 51% and at least 46% in each category using novel schedulers. The SSN Scheduler performs the worst, even in CAT 1 and 2 mean age reduction, while the integer programs are slightly behind the relaxed models.

5.3.4 Max Age

Most of the schedulers perform the same, providing a very low max age for CAT 1's and sacrificing CAT 4 and 5's max age. However, SSN Scheduler performs the absolute worst compared to the other models, since it is alone in not leveraging a centralized scheduler. Up to 9% decrease in maximum age is seen in novel schedulers over all RSO's with at least 22% decrease in CAT 1,2, and 5 RSO's.

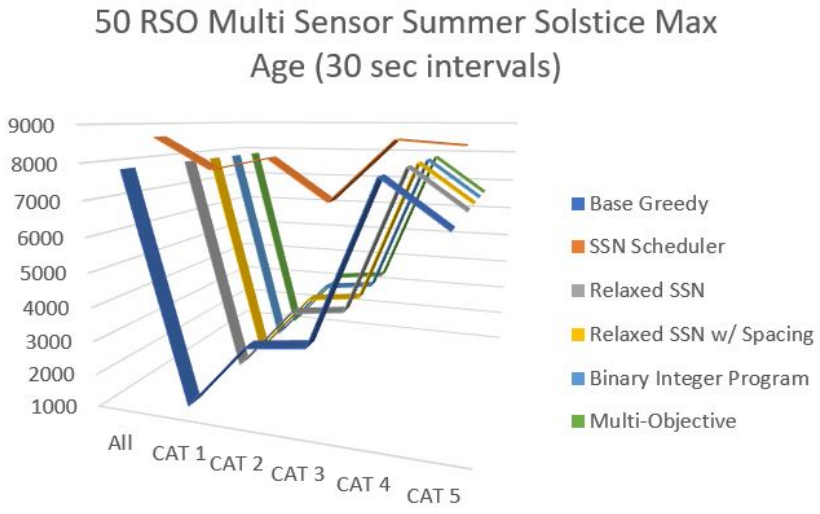

Figure 61. 50 RSO Summer Solstice Max Age

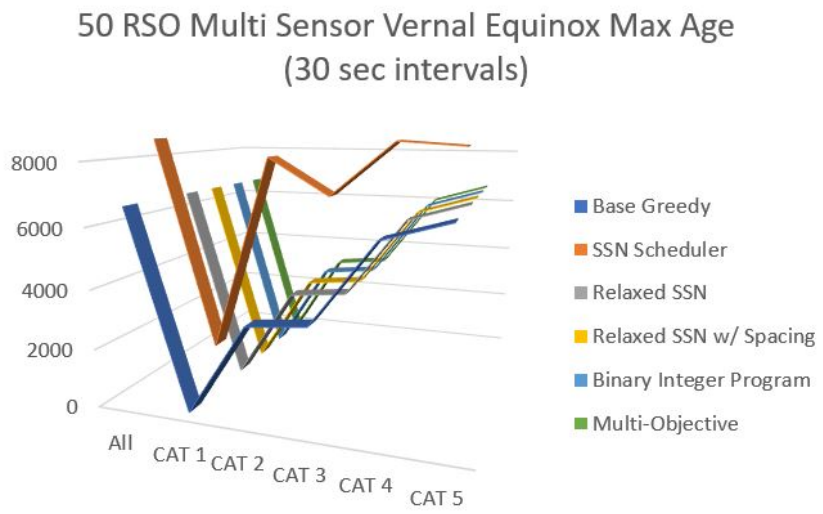

Figure 62. 50 RSO Vernal Equinox Max Age

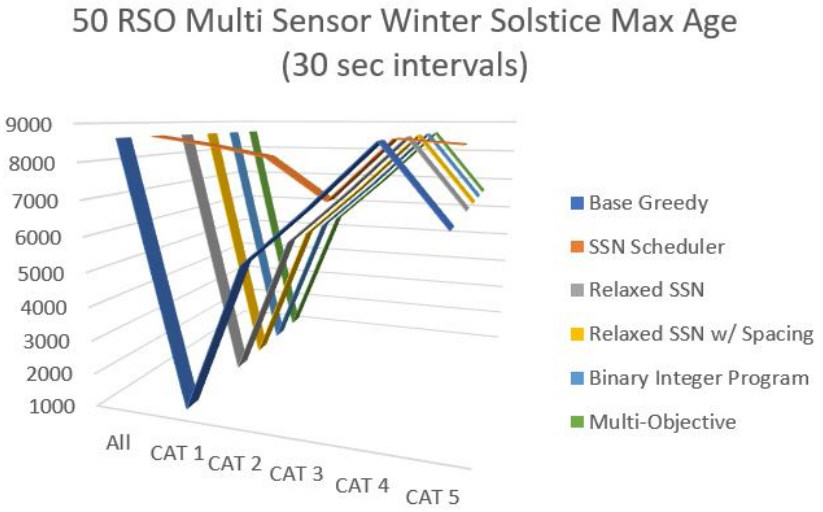

Figure 63. 50 RSO Winter Solstice Max Age

5.3.4.1 SSN Scheduler Model

The SSN Scheduler performs the worse in reducing max age, including CAT 1's and 2's by at least 1500 intervals and by as large as 7000 intervals.

5.3.4.2 All Other Models

All other models performed the same, allowing CAT 4's to have the worst max age during each solstice and trading with CAT 5 being worst during Vernal Equinox. Cat 1's max age were kept at minimums just above 1000 intervals during the solstices and nearly zeroed during the Vernal Equinox.

5.3.4.3 Conclusion

Max age is significantly worse using the SSN Scheduler model. Up to 9% decrease in maximum age is seen in novel schedulers over all RSO's with at least 22% decrease in CAT 1,2, and 5 RSO's. With all other models, max age is the same with each model and each model besides the SSN Scheduler brings the max age of CAT 1's down nearly to zero during Vernal Equinox.

5.3.5 Run Times

Both the Binary Integer Program Model and the Multi-objective Binary Integer Program Model take significantly longer to execute than the other models. Both scripts take approximately 10-15 times as long.

5.3.5.1 Base Model Greedy

The Base Model Greedy script runs the fastest of all schedulers.

5.3.5.2 SSN Scheduler Model

The SSN Scheduler runs at comparable speed to the relaxed models, but is slower than the greedy model.

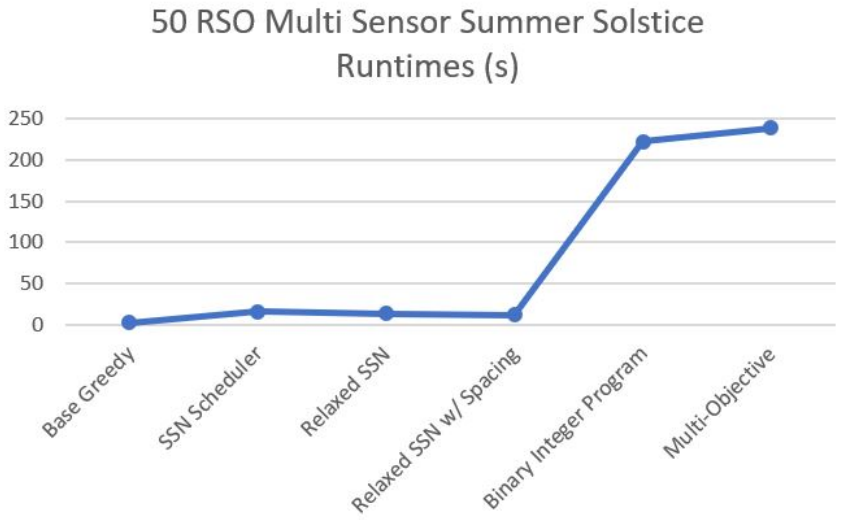

Figure 64. 50 RSO Summer Solstice Run Times

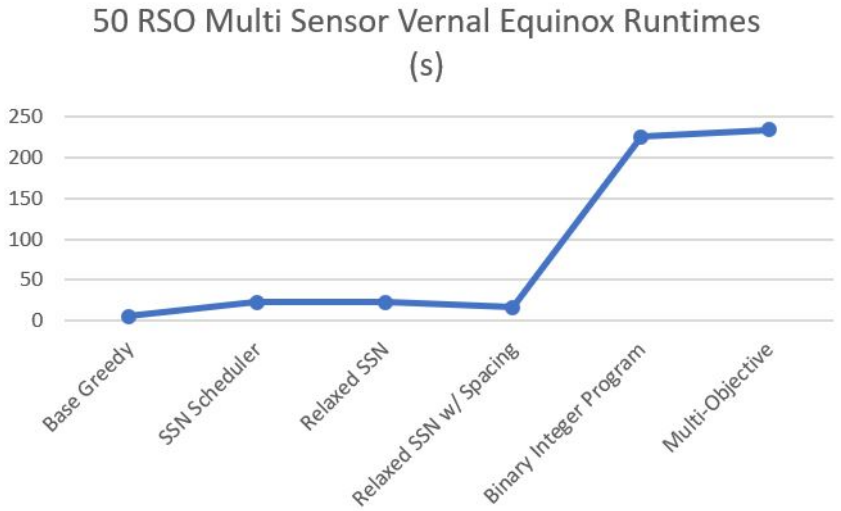

Figure 65. 50 RSO Vernal Equinox Run Times

5.3.5.3 Relaxed SSN Scheduler Model

The first relaxed model scheduler runs slightly faster than the SSN Scheduler on both solstices.

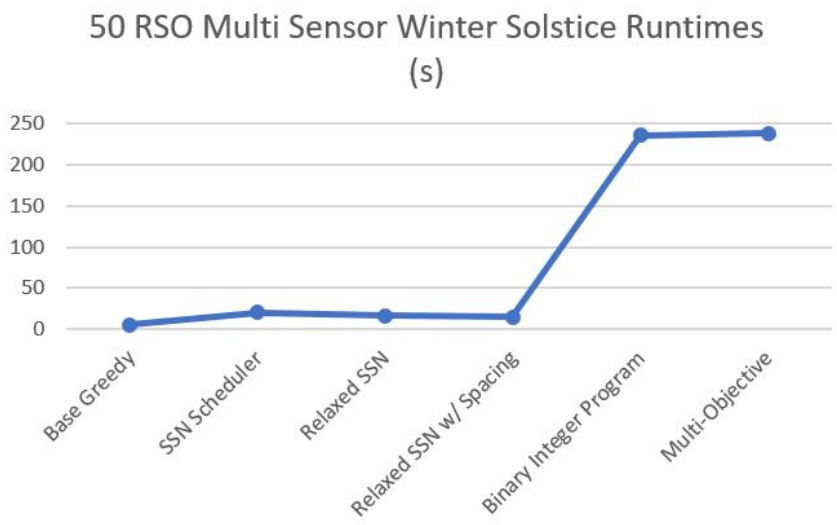

Figure 66. 50 RSO Winter Solstice Run Times

5.3.5.4 Relaxed SSN Scheduler with Spacing Model

The second relaxed model runs even faster than the first relaxed model, beating the SSN Scheduler, but not quite near the greedy model.

5.3.5.5 Binary Integer Program Model

The Binary Integer Program runs significantly longer than the heuristic style schedulers. On each day, the Binary Integer Program Model runs approximately 15-20 times as long to create a solution.

5.3.5.6 Multi-Objective Binary Integer Program Model

The Multi-Objective Binary Integer Program Model runs significantly longer than the heuristic style schedulers and runs about ten seconds slower than the Binary Integer Program Model.

5.3.5.7 Conclusion

The Base Model Greedy scheduler runs faster than all others. The SSN Scheduler and both relaxed models follow close behind, especially when compared to that of the integer programs. The integer programs take 15-20 times as long as the other schedulers to develop a solution.

5.3.6 Conclusion

Overall, the SSN Scheduler is shown to perform significantly worse than other schedulers in several areas. In addition, it doesn't improve the areas of its focus - CAT 1 and CAT 2 RSO's likely because each sensor is viewing the same objects. Novel schedulers show at least 10% improvement in overall threshold numbers, 51% decrease in mean age, and up to 9% decrease in maximum age of all RSO's. The relaxed models provide lower mean ages, while the integer programs provide more higher priority RSO's meeting threshold. Additionally, the integer programs have the downside of taking significantly more time to process. The fact that both relaxed models have run times consistent with the SSN Scheduler yet both are centrally scheduling ground assets is strong support for consideration of a centralized scheduler. This is especially true as we consider the performance improvements centralized schedulers exhibit in sections 5.3.2, 5.3.3, 5.3.4, and 5.3.5.

5.4 Summary

Chapter V reviewed multi-sensor changes to the methodology of the six different models introduced in Chapter III. Then, the chapter covered the results of analysis using multiple sensors. Tables with result data can be found in Appendix B. Chapter VI covers conclusions for this research and recommendations for future study.

VI. Conclusions and Future Research

6.1 Overview

This chapter summarizes the research addressed in this thesis. It begins by answering the investigative question posed in Chapter I. Next, conclusions drawn from this research are addressed along with a presentation of the significance of this research. Finally, recommendations for future research, model improvements, and potential follow-on efforts are addressed.

6.2 Conclusions of Research

SSA is one of the most crucial missions faced by the DoD. Scheduling SSN ground and space based assets across a variety of orbit and mission types is a difficult problem becoming more complex due to increased national dependence on space assets and ever growing congestion from national, foreign, and commercial entities. The necessity for a means to efficiently track objects and debris is vital to ensuring the US maintains dominance in the contested space domain. The research effort addressed in this thesis covers the problem of scheduling SSN ground assets in support of the SSA mission across a variety of orbit types. The development of a centralized multi-sensor, multi-mission scheduling model is necessary to ensure all assets are working harmoniously. This research developed two relaxed heuristic schedulers based on the guidance provided by the 2004 SD 505-1 and the 18th Space Control Squadron and compared them to both a greedy heuristic base model from prior research, and a model reflecting the guidance laid out in the 2004 SD 505-1 SSN Scheduler. Additionally, two integer program schedulers were developed for comparison. The schedulers were simulated on three sample sizes of RSO's, on three different days of the year, with both a single sensor and multiple sensors. The metrics evaluated were total

observations made by priority category, total number of RSO's meeting observation threshold requirements, mean and maximum time between observations, and the run time required for the scheduling scripts. The research identified areas where the SSN Scheduler Model were lacking compared to the novel schedulers and areas where each novel scheduler excelled and were deficient.

Investigative Questions:

- Can a scheduling method be developed which outperforms the scheduler represented in 2004 SD 505-1 at reducing mean and maximum time between observation for a population of RSO's while meeting RSO observation thresholds?
- Can a novel scheduler be developed that improves upon the scheduler used in prior research with roughly same processing requirements?

On the single sensor scale, both relaxed constraint schedulers and both integer program schedulers were shown to provide an increase in observations of non-CAT 1 RSO categories at the expense of reducing observations of CAT 1 RSO's compared to the SSN Scheduler Model. Each novel scheduler demonstrated the ability to maintain the same or similar level of observation threshold retention in CAT 1, while increasing observation threshold gains in other categories. Doing so reduced the mean time between observations for all non-CAT 1 categories with minimal impact to CAT-1 mean time between observations. The research also demonstrated the novel schedulers' ability to impact the maximum age of observation via slightly improved max age in certain categories. Novel schedulers improve overall threshold numbers by 3%-20% and decrease mean age up to 12%-33% overall compared to the SSN Scheduler Model. The processing time required for relaxed constraint schedulers was indifferent from the processing time of the SSN Scheduler Model, demonstrating the novel schedulers improve upon the Base Model Greedy Scheduler used in prior theses with little additional processing time. Implementing integer program schedulers required

a significant increase in both processing time and hardware resources as compared to heuristic schedulers.

On the multi-sensor scale, both relaxed constraint schedulers and both integer programs ran as centralized schedules and demonstrated the ability to increase total observations of non-CAT 1 RSO's while maintaining or improving the number of RSO's meeting observation thresholds in each priority category as compared to the SSN Scheduler Model. Additionally, the integer programs showed an increase in some non-CAT 1 higher priority RSO's over lower priority targets during scheduling as compared to the relaxed constraint schedulers. Implementing the novel schedulers in the multi-sensor scale showed dramatic improvement in mean age of RSO sample population over the SSN Scheduler Model in both mean time between observation and maximum age of time between observation. Novel schedulers show at least 10% improvement in overall threshold numbers, 51% decrease in mean age, and up to 9% decrease in maximum age of all RSO's. Most notably, those improvements did not come at much computational cost as processing time was similar to single sensor results for both relaxed models, again demonstrating improvement on the Base Model Greedy Scheduler with little additional processing time. Integer programs did demonstrate an expected significant increase in processing time and hardware resources, making them a less attractive option compared to either of the relaxed scheduler models.

6.3 Significance of Research

The research demonstrated the feasibility and advantage of expanding to a centrally scheduled and cooperative sensor network. Novel heuristics were run on such a network and provided superior results versus the current models while maintaining computational tractability.

6.4 Recommendations for Future Research

This section discusses some ideas for potential follow-on work and future research opportunities. There are various ways of modifying this research to better suit more specific or generic missions.

1. Expansion of the RSO population
2. Inclusion of a variety of sensors
3. Longer scenario duration
4. Dynamic priority list
5. Changes to the target list
6. Orbit or mission based interval spacing
7. Incorporate alternate measures
8. Alterations to sensor capabilities
9. Combine novel schedules with architecture optimization
10. Explore alternative or combined schedulers

The recommendations are explained below:

1. This research limited the RSO population when comparing to integer program schedulers because of the RAM limitations on the hardware available to run the software. During this research, installation of the PuLP software was requested on more advanced hardware, but was not completed in time. Once installation is complete, exploring the results of larger populations could show more dramatic results. Additionally, AFRL DSRC's HPC could potentially be utilized

as the script was intentionally written in Python for compatibility with Linux based systems. RSO population changes could include RSO's being launched or changing orbit and sensor changes could involve modeling various real world sensors and missions. Modeling the full SSN inventory with the whole RSO population could show the true impact novel schedulers may bring to the SSN.

2. This research mimicked the capabilities and locations of the GEODSS sensors of the SSN. Inclusion of different sensors with varying ranges of limitations such as range, image type, imaging requirements, would more accurately model a large scale sensor network. These other sensors could be explored individually, as smaller networks, include space-based sensors, or include hypothetical future sensors, such as Space Fence.

3. This research imitated prior research in developing schedules for a single 24-hour period. Future research could expand the scenario timeline in order to determine the time required for metrics such as mean and maximum time of observations to converge.

4. A static random priority list provided at the start of the scenario day was used, however, as an RSO's time between observations increases over time, the priority level is increased. Incorporating a dynamic priority list into the scheduling algorithm would more accurately model the 2004 SD 505-1 SSN Scheduler Model. Alternatively, adjusting the sample population priority numbers to determine at what point saturation in higher priority categories occurs in the SSN Scheduler Model would show how close current operations are to reaching the limits of sensor capabilities.

5. The current target list at the beginning of a scenario is static. Changing the target list to adapt to new high priority targets such as space launch vehicles,

assets which have changed orbit, or newly discovered RSO's would imitate a dynamically changing environment.

6. The Relaxed Constraints with Spacing Model scheduler included a set spacing interval of sixty 30-second intervals. Adapting the intervals to include different orbital spacing based on the orbital period or type of sat (LEO, GEO, etc.) would more accurately reflect high-confidence orbits [24].

7. This research utilized SNR values to determine whether an RSO was visible or not. Having a large signal may not necessarily mean RSO features are distinguishable, therefore alternate methods such as visual magnitude may be a desired variable to consider instead of SNR. It also utilized time between observations as a measure of uncertainty, the larger the time between observations, the larger the assumption that an observation would be required. Incorporating Gauss equations to calculate uncertainty of orbits could be an alternate method to determine requirements for observation.

8. When an observation is made on a target RSO, only that one specific RSO is considered observed even if there are multiple RSO's in the sensors FOV. Other RSO's may be clustered in view of observation and the sensors instantaneous field of view may be large enough to capture multiple targets at once. An updated scheduler might recognize these clusters as multiple RSO's and collect multiple observations in a single observation window, significantly improving the performance of the sensor and SSN. Similarly, the observation window may be decreased or made dynamic to enable more observations or allot additional time to higher priority targets. RSO's closer together require less slew and settle time and an updated scheduler could take advantage of these time savings.

9. This research was based on prior thesis work using a Genetic Algorithm to

optimize SSA architectures, work that has garnered lots of attention and interest from various stakeholders. The optimization involves a multi-objective evaluation of each architecture with the time between observations being one of the evaluation metrics. That research can be replicated using the schedulers developed here to determine changes to architecture requirements.

10. This research explored greedy based heuristics, integer program scheduling, and multi-objective optimization. Additional schedulers could be compared to include stable marriage optimization, Linear Program Relaxation, simplified binary integer programs using preprocessing of known data to remove variables or constraints, various machine learning or distributed Q networks, Genetic Algorithm schedulers.

6.5 Summary

This chapter summarized the overall thesis objective and provided answers to the research question presented in Chapter I. Several novel schedulers were developed and compared to an adapted SSN scheduler based on 2004 SD 505-1. Various metrics were compared to include total observations, total number of RSO's meeting observation threshold requirements, mean and maximum time between observations and total script processing time. Suggestions for future research were outlined to aid in the progression of this thesis. SSA is a unique problem with both national security concerns and commercial implications. As launch costs decrease and more nations gain space assets in different orbital regimes, the space domain becomes more congested and contested. The necessity to identify and track RSO's in orbit and positively attribute actions of spacecraft becomes more important as human involvement in space progresses. With more efficient, centralized, schedulers, sensor assets become more capable of keeping up with the vastly growing spacecraft and debris collection in

orbit, generating more detailed and effective data for analysis, and communicate accurate real time representations of the current space environment to decision makers providing crucial and innovative tools for future SSA policy.

Appendix A. Analysis Python Code

The following contains Python code scripts for data generation and each scheduler.

1.1 One Sensor Python Code

1.1.1 Data Generation

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script connects with STK and generates access reports for one ground
sensor with all RSO's populated.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import signal
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from itertools import izip as izip, count
import re
import platform
import commands as c
import random

from Targets import tgtList

TS=86400 #total number of seconds in a 24 hour period
repTS=30 #Length of the report timesteps in seconds

print "Started"
staT=time.time()
plat=platform.system()
commands=[]
repCommands=[]
global numInst
dtStart = '21 Jun 2019 00:00:00' #Start date & time
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time

if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object used to pass info from Python to STK
    for res in socket.getaddrinfo
        (HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
```

1

```
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

numGnd1=1
#numGnd2=1
#numGnd3=1
numInst=1

 ###########################################################
numTgt=200
numSensors=1#3 #sum([numGnd1,numGnd2,numGnd3])
if numSensors!=0:

    #%%
    repID='Rep'+str(numInst)
    rstID='RST'+str(numInst)
    #%% Windows directories
    cwd=os.getcwd()
    workPath=cwd
        #Here must be changed to the proper directorties
    rstPath = os.path.join(workPath,'RSTfiles_Jun')
    curRstPath = os.path.join(rstPath,rstID)
    if os.path.exists(curRstPath):
       rstGlob = os.path.join(curRstPath,'*.rst')
       files=glob.glob(rstGlob)
        #for f in files:
            #os.remove(f)
    else:
        try:
            os.mkdir(rstPath)
        except:
            pass
        os.mkdir(curRstPath)
   rstPath=curRstPath
       #Here must be changed to the proper directorties
   repPath = os.path.join(workPath,'Reports_Jun')
   curRepPath = os.path.join(repPath,repID)
    if os.path.exists(curRepPath):
        repGlob = os.path.join(curRepPath,'*.txt')
        files=glob.glob(repGlob)
        #for f in files:
            #os.remove(f)
    else:
        try:
            os.mkdir(repPath)
        except:
```

```
        pass
    os.mkdir(curRepPath)
repPath=curRepPath
os.chdir(rstPath)

#%% Write moon phase report template
f=open('MoonPhase.rst','w')
f.write('stk.v.11.0\nWrittenBy    STK_v11.2.0\n \nBEGIN ReportStyle\n\n')
f.write('BEGIN ClassId\n    Class    Scenario\nEND ClassId\n\n')
f.write('BEGIN Header\n')
f.write('    StyleType    0\n    Date    Yes\n    Name    Yes\n    IsHidden    No\r
f.write('    DescShort    No\n    DescLong    No\n    YLog10    No\n    Y2Log10
f.write('    YUseWholeNumbers    No\n    Y2UseWholeNumbers    No\n    VerticalGridL
f.write('    HorizontalGridLines    No\n    AnnotationType    Spaced\n    NumAnnota
f.write('    NumAngularAnnotations    5\n    ShowYAnnotations    Yes\n    Annotatic
f.write('    BackgroundColor    #ffffff\n    ForegroundColor    #000000\n    Viewab
f.write('    RealTimeMode    No\n    DayLinesStatus    1\n    LegendStatus    1\n
f.write('BEGIN PostProcessor\n    Destination    0\n    Use    0\n    Destination
f.write('    Destination    2\n    Use    0\n    Destination    3\n    Use    0\nEN
f.write('    NumSections    1\nEND Header\n\n')
f.write('BEGIN Section\n    Name    Section 1\n    ClassName    Scenario\n    NameI
f.write('    ExpandMethod    0\n    PropMask    2\n    ShowIntervals    No\n    Num
f.write('BEGIN Line\n    Name    Line 1\n    NumElements    2\n\n')
f.write('BEGIN Element\n    Name    Time\n    IsIndepVar    Yes\n')
f.write('    IndepVarName    Time\n    Title    Time\n    NameInTitle    No\n    Se
f.write('    Type    Moon LunarPhase\n    Element    Time\n    SumAllowedMask    0\
f.write('    DataType    0\n    UnitType    2\n    LineStyle    0\n    LineWidth
f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #000000\n    Fi
f.write('    UseScenUnits    Yes\nEND Element\n\n')
f.write('BEGIN Element\n    Name    Angles-Moon LunarPhase-Angle\n    IsIndepVar
f.write('    IndepVarName    Time\n    Title    Angle\n    NameInTitle    Yes\n
f.write('    Type    Moon LunarPhase\n    Element    Angle\n    SumAllowedMask    1
f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n    LineWidth
f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #000000\n    Fi
f.write('    UseScenUnits    Yes\n')
f.write('END Element\nEND Line\nEND Section\n\n')
f.write('BEGIN LineAnnotations\nEND LineAnnotations\nEND ReportStyle')
f.close()
#%% Write angle report templates for GBT
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            f=open('Tgt_'+str(i)+'_from_Gnd'+str(m)+'_Angles.rst','w')
            f.write('stk.v.11.0\nWrittenBy    STK_v11.2.0\n \nBEGIN ReportStyle\n\r
            f.write('BEGIN ClassId\n    Class    Satellite\nEND ClassId\n\n')
            f.write('BEGIN Header\n')
            f.write('    StyleType    0\n    Date    Yes\n    Name    Yes\n    IsHi
            f.write('    DescShort    No\n    DescLong    No\n    YLog10    No\n
            f.write('    YUseWholeNumbers    No\n    Y2UseWholeNumbers    No\n    V
            f.write('    HorizontalGridLines    No\n    AnnotationType    Spaced\n
            f.write('    NumAngularAnnotations    5\n    ShowYAnnotations    Yes\n
            f.write('    BackgroundColor    #ffffff\n    ForegroundColor    #000000
            f.write('    RealTimeMode    No\n    DayLinesStatus    1\n    LegendSta
            f.write('BEGIN PostProcessor\n    Destination    0\n    Use    0\n    D
            f.write('    Destination    2\n    Use    0\n    Destination    3\n
```

```
            f.write('    NumSections    1\nEND Header\n\n')
            f.write('BEGIN Section\n    Name    Section 1\n    ClassName    Satelli
            f.write('    ExpandMethod    0\n    PropMask    2\n    ShowIntervals
            f.write('BEGIN Line\n    Name    Line 1\n    NumElements    3\n\n')
            f.write('BEGIN Element\n    Name    Time\n    IsIndepVar    Yes\n')
            f.write('    IndepVarName    Time\n    Title    Time\n    NameInTitle
            f.write('    Type    PhaseAngle_Gnd'+str(m)+'\n    Element    Time\n
            f.write('    DataType    0\n    UnitType    2\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\nEND Element\n\n')
            f.write('BEGIN Element\n    Name    Angles-PhaseAngle_Gnd'+str(m)+'-Ang
            f.write('    IndepVarName    Time\n    Title    Solar Phase Angle\n
            f.write('    Type    PhaseAngle_Gnd'+str(m)+'\n    Element    Angle\n
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\nEND Element\n')
            f.write('BEGIN Element\n    Name    Angles-LunarPhaseAngle_Gnd'+str(m)+
            f.write('    IndepVarName    Time\n    Title    Lunar Phase (Obs) Angle
            f.write('    Type    LunarPhaseAngle_Gnd'+str(m)+'\n    Element    Angl
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\n')
            f.write('END Element\nEND Line\nEND Section\n\n')
            f.write('BEGIN LineAnnotations\nEND LineAnnotations\nEND ReportStyle')
            f.close()
#%% Write zenith angle report templates for GBT
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            f=open('Gnd'+str(m)+'_to_Tgt_'+str(i)+'_ZenithAngles.rst','w')
            f.write('stk.v.11.0\nWrittenBy    STK_v11.2.0\n \nBEGIN ReportStyle\n\r
            f.write('BEGIN ClassId\n    Class    Facility\nEND ClassId\n\n')
            f.write('BEGIN Header\n')
            f.write('    StyleType    0\n    Date    Yes\n    Name    Yes\n    IsHi
            f.write('    DescShort    No\n    DescLong    No\n    YLog10    No\n
            f.write('    YUseWholeNumbers    No\n    Y2UseWholeNumbers    No\n    \
            f.write('    HorizontalGridLines    No\n    AnnotationType    Spaced\n
            f.write('    NumAngularAnnotations    5\n    ShowYAnnotations    Yes\n
            f.write('    BackgroundColor    #ffffff\n    ForegroundColor    #000000
            f.write('    RealTimeMode    No\n    DayLinesStatus    1\n    LegendSta
            f.write('BEGIN PostProcessor\n    Destination    0\n    Use    0\n    D
            f.write('    Destination    2\n    Use    0\n    Destination    3\n
            f.write('    NumSections    1\nEND Header\n\n')
            f.write('BEGIN Section\n    Name    Section 1\n    ClassName    Facilit
            f.write('    ExpandMethod    0\n    PropMask    2\n    ShowIntervals
            f.write('BEGIN Line\n    Name    Line 1\n    NumElements    3\n\n')
            f.write('BEGIN Element\n    Name    Time\n    IsIndepVar    Yes\n')
            f.write('    IndepVarName    Time\n    Title    Time\n    NameInTitle
            f.write('    Type    LunarZenith\n    Element    Time\n    SumAllowedMa
            f.write('    DataType    0\n    UnitType    2\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\nEND Element\n\n')
            f.write('BEGIN Element\n    Name    Angles-LunarZenith-Angle\n    IsInc
            f.write('    IndepVarName    Time\n    Title    Lunar Zenith Angle\n
            f.write('    Type    LunarZenith\n    Element    Angle\n    SumAllowedM
```

```
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #00
            f.write('    UseScenUnits    Yes\nEND Element\n')
            f.write('BEGIN Element\n    Name    Angles-TargetZenith_Gnd'+str(m)+'_t
            f.write('    IndepVarName    Time\n    Title    Target Zenith Angle\n
            f.write('    Type    TargetZenith_Gnd'+str(m)+'_to_Tgt_'+str(i)+'\n
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #00
            f.write('    UseScenUnits    Yes\n')
            f.write('END Element\nEND Line\nEND Section\n\n')
            f.write('BEGIN LineAnnotations\nEND LineAnnotations\nEND ReportStyle')
            f.close()

#%% Change directory back to working directory
os.chdir(workPath)
#%% Create Scenario
scenName='Thesis'
concontrol='ConControl / VerboseOn'
commands.append(concontrol) #Please note this script utilizes commands.append not s
unload='Unload / *'
commands.append(unload)
newScen= 'New / Scenario '+str(scenName)
commands.append(newScen)
setTimePeriodStr = 'SetTimePeriod * "' + str(dtStart)+ '" "' + str(dtStop)
    +'"'
commands.append(setTimePeriodStr)
#%% Create Moon Phase Angle
angStr='VectorTool * CentralBody/Moon Create Angle LunarPhase "Between Vectors" "Ce
commands.append(angStr)
#%% Create Equally Spaced Target Sats
for n in range(1,numTgt+1):
    tgt = str(n)
    newTgt = 'New / */Satellite Tgt_' + str(tgt)
    commands.append(newTgt)
    epoch=tgtList[n-1][0]
    semiMajAxis=tgtList[n-1][1]
    ecc=tgtList[n-1][2]
    inc=tgtList[n-1][3]
    argOfPerigee=tgtList[n-1][4]
    RAAN=tgtList[n-1][5]
    meanAnom=tgtList[n-1][6]
    setStateTgt = ('SetState */Satellite/Tgt_' + str(tgt)
        +' Classical J2Perturbation \"'+ str(dtStart)+ '\" \"'+str(dtStop)
        +' \" '+str(TS)+' J2000 \"'+str(epoch)+'\"' +str(semiMajAxis)
        +' '+ str(ecc) +' '+ str(inc) +' '+str(argOfPerigee) +' '
        + str(RAAN) +' '+str(meanAnom) )
    commands.append(setStateTgt)
    tgtDirLighting="SetConstraint */Satellite/Tgt_"+str(tgt)
        +" Lighting DirectSun"
    commands.append(tgtDirLighting)

#%% Create Ground-Based Telescopes
print 'Targets created'
loc=[[33.8200 , -106.6600, 1403]]
#this is iterating through the above list to set the facilities locations and the c
```

```
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        telStr='New / */Facility Gnd'+str(m)
        commands.append(telStr)
        locStr="SetPosition */Facility/Gnd"+str(m)+" Geodetic "
            + str(loc[m-1][0]) + ' ' + str(loc[m-1][1]) + ' '
            +str(loc[m-1][2])
        commands.append(locStr)
        solarExc= "SetConstraint */Facility/Gnd"+str(m)
            +" LOSSunExclusion 40"
        commands.append(solarExc)
        lunarExc="SetConstraint */Facility/Gnd"+str(m)
            +" LOSLunarExclusion 10"
        commands.append(lunarExc)
        lighting="SetConstraint */Facility/Gnd"+str(m)+" Lighting Umbra"
        commands.append(lighting)
        elevationAngle="SetConstraint */Facility/Gnd"+str(m)
            +" ElevationAngle Min 15.0"
        commands.append(elevationAngle)

#%% Create "To Sensor" vector and Phase Angle for each target/sensor pair.
#Also create facility-to-target vector and target zenith angle for GBTs.
print 'GBTs created'
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            vecStr1='VectorTool * Satellite/Tgt_'+str(i)
                +' Create Vector ViewVector_Gnd'+str(m)
                +' "Displacement" "Satellite/Tgt_'+str(i)
                +' Center" "Facility/Gnd'+str(m)+' Center"'
            commands.append(vecStr1)
            vecStr2='VectorTool * Facility/Gnd'+str(m)
                +' Create Vector F2T_Gnd'+str(m)+'_to_Tgt_'
                +str(i)+' "Displacement" "Facility/Gnd'+str(m)
                +' Center" "Satellite/Tgt_'+str(i)+' Center"'
            commands.append(vecStr2)
            angStr1='VectorTool * Satellite/Tgt_'+str(i)
                +' Create Angle PhaseAngle_Gnd'+str(m)
                +' "Between Vectors" "Satellite/Tgt_'+str(i)
                +' ViewVector_Gnd'+str(m)+'" "Satellite/Tgt_'+str(i)
                +' Sun"'
            commands.append(angStr1)
            angStr2='VectorTool * Facility/Gnd'+str(m)
            +' Create Angle TargetZenith_Gnd'+str(m)+'_to_Tgt_'+str(i)
            +' "Between Vectors" "Facility/Gnd'+str(m)+' F2T_Gnd'+str(m)
            +'_to_Tgt_'+str(i)+'" "Facility/Gnd'+str(m)+' Zenith"'
            commands.append(angStr2)

#%% Create Lunar zenith angle for each GBT and phase angle for each
#target/GBT pair
print 'to sensor vector and phase angle created'
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            angStr1='VectorTool * Satellite/Tgt_'+str(i)
```

```
                +' Create Angle LunarPhaseAngle_Gnd'+str(m)
                +' "Between Vectors" "Satellite/Tgt_'+str(i)
                +' ViewVector_Gnd'+str(m)+'" "Satellite/Tgt_'+str(i)
                +' Moon"'
            commands.append(angStr1)
        angStr2='VectorTool * Facility/Gnd'+str(m)
            +' Create Angle LunarZenith "Between Vectors" "Facility/Gnd'
            +str(m)+' Zenith" "Facility/Gnd'+str(m)+' Moon"'
        commands.append(angStr2)
#%% Load report styles
print 'Lunar zenith angle created'
os.chdir(rstPath)
loadStrPath = os.path.join(rstPath,'MoonPhase.rst')
moonPath = loadStrPath
loadStr='ReportStyle * Load "'+  str(loadStrPath)  +'"'
commands.append(loadStr)
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            loadStr1Path=os.path.join(rstPath, 'Tgt_'+str(i)
                +'_from_Gnd'+str(m)+'_Angles.rst')
            loadStr1='ReportStyle * Load "'+  str(loadStr1Path) +'"'
            commands.append(loadStr1)
            loadStr2Path=os.path.join(rstPath, 'Gnd'+str(m)+'_to_Tgt_'
                +str(i)+'_ZenithAngles.rst')
            loadStr2='ReportStyle * Load "'+  str(loadStr2Path) +'"'
            commands.append(loadStr2)

#%% Compute access and create access reports for each target/sensor pair
config="ExportConfig / Connection Headers None KeepReportLines Off ShowStartStop Of
repCommands.append(config)
repStrPath =os.path.join(repPath,'MoonPhase.txt')
repStr='ReportCreate * Type Export Style "'+str(moonPath)+'" File "'
    +  str(repStrPath)+'" TimeStep 60'
repCommands.append(repStr)
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            repStrPath=os.path.join(repPath, 'Tgt_'+str(i)+'_from_Gnd'
                +str(m)+'_AccessRep.txt')
            repStr='ReportCreate */Satellite/Tgt_'+str(i)
                +' Type Export Style "Access" File "'+  str(repStrPath)
                +'" AccessObject */Facility/Gnd'+str(m)
            repCommands.append(repStr)

#%% Send commands to STK
print 'access reports created'
#Keep an eye for different sections that use the plat=='windows'
#and plat!='windows'
if plat=='Windows':
    repCommands.append('Animate * Reset *')
    for x in commands:
        try:
            s.send(str(x)+'\n')
        except socket.error, e:
```

```
            if e.args[0]==10035:
                flag=0
                while flag==0:
                    time.sleep(1)
                    try:
                        s.send(str(x)+'\n')
                        flag=1
                    except:
                        pass
            else:
                print e
                break
   time.sleep(1)
    for y in repCommands:
        try:
            s.send(str(y)+'\n')
        except socket.error, e:
            if e.args[0]==10035:
                flag=0
                while flag==0:
                    time.sleep(1)
                    try:
                        s.send(str(y)+'\n')
                        flag=1
                    except:
                        pass
            else:
                print e
                break

numReps=numSensors*numTgt
#%% Wait for Access reports to show up
os.chdir(repPath)
repCount=0
while repCount<numReps:
    for m in range(1,2):#4):
        if (eval('numGnd'+str(m)))>0:
            fileName='Tgt_'+str(numTgt)+'_from_Gnd'+str(m)+'_AccessRep.txt'
            if os.path.exists(fileName):
                repCount+=1
            else:
                time.sleep(.1)

   #%% Check to make sure an access occured before creating angle reports
print "All Access Reps found for Instance "+str(numInst)
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            accessName = 'Tgt_'+str(i)+'_from_Gnd'+str(m)+'_AccessRep.txt'
            accessArray = [[str(x) for x in line.strip().split(',')]
                for line in open(accessName, 'r')]
            lenAccess=len(accessArray)
            if lenAccess!=0:
                repStrPath=os.path.join(repPath,'Gnd'+str(m)
                    +'_to_Tgt_'+str(i)+'_ZenithAngles.txt')
```

```
                repStr='ReportCreate */Facility/Gnd'+str(m)
                    +' Type Export Style "Gnd'+str(m)+'_to_Tgt_'
                    +str(i)+'_ZenithAngles" File "'+str(repStrPath)
                    +'" TimeStep '+str(repTS)
                repStr1Path=os.path.join(repPath,'Tgt_'+str(i)
                    +'_from_Gnd'+str(m)+'_AngleRep.txt')
                repStr1='ReportCreate */Satellite/Tgt_'+str(i)
                    +' Type Export Style "Tgt_'+str(i)+'_from_Gnd'+str(m)
                    +'_Angles" File "'+str(repStr1Path)+'" TimeStep '
                    +str(repTS)
                commands.append(repStr)
                commands.append(repStr1)

#%% Create AER reports for each target/sensor pair
for m in range(1,2):#4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            repStrPath=os.path.join(repPath,'Tgt_'+str(i)+'_from_Gnd'
                +str(m)+'_AERRep.txt')
            repStr='ReportCreate */Facility/Gnd'+str(m)
                +' Type Export Style AER File "'+str(repStrPath)
                +'" AccessObject */Satellite/Tgt_'+str(i)+' TimeStep '
                +str(repTS)
            commands.append(repStr)

#%% Send commands to STK
print 'AER reports created for each target/sensor pair'
if plat=='Windows':
    commands.append('Animate * Reset *')
    for x in commands:
        try:
            s.send(str(x)+'\n')
        except socket.error, e:
            if e.args[0]==10035:
                flag=0
                while flag==0:
                    time.sleep(1)
                    try:
                        s.send(str(x)+'\n')
                        flag=1
                    except:
                        pass
            else:
                print e
                break

os.chdir(repPath)
repCount=0
while repCount<numReps:
    for m in range(1,2):#4):
        if (eval('numGnd'+str(m)))>0:
            fileName='Tgt_'+str(numTgt)+'_from_Gnd'+str(m)+'_AERRep.txt'
            if os.path.exists(fileName):
                repCount+=1
            else:
```

```
                    time.sleep(.1)


    print "STK finished for Instance "+str(numInst)
    stopT=time.time()
    runTime=(stopT-staT)
    print 'Runtime: ' +str(runTime)
    time.sleep(1)

#%% Clean Up Access reports with bad phase angles
    ObservationDuration=repTS
    print "Instance "+str(numInst)+" done."
```

1.1.2 Base Greedy Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
#import pty
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=190

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
```

```
probCat4E=0.05
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]

staT=time.time()
plat=platform.system()

#Start date & time remember to match these up with the same dates as when you
#generated the data
dtStart = '21 Jun 2019 00:00:00'
#Stop date & time
dtStop= '22 Jun 2019 00:00:00'
#This indicates what num trial is being run when this script is being
#executed, must match other scripts
trial_num = 'Trial_10'
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    # s is a socket object that we will use to pass info from our Python
    #program to STK
    s = None
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC,
            socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
```

```
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])

repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

print "Instance "+str(numInst)
print repLocs
print numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,
        '%d %b %Y %H:%M:%S')-datetime.datetime.strptime(dtStart,
        '%d %b %Y %H:%M:%S'))/86400
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
```

```
Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
    #400nm to 800nm (min to max nm)
numObservers=3
UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
AllObservationIntervals=[]
for g in xrange(0,Intervals):#Creates the list of time steps
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)

PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#This section assigns priority categories to each sat in list. assignment
#is based on probability of each category using given constants. will
#assign a number 1-5 and subcategory 1-5 (A-E) as a decimal
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A)*100:
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A+probCat2B)*100:
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A+probCat2B+probCat2C)*100:
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A+probCat2B+probCat2C+probCat2D)*100:
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E)*100:
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A)*100:
```

```
    tempCat=3.1
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B)*100:
    tempCat=3.2
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C)*100:
    tempCat=3.3
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D)*100:
    tempCat=3.4
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E)*100:
    tempCat=3.5
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A)*100:
    tempCat=4.1
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B)*100:
    tempCat=4.2
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
    +probCat4C)*100:
    tempCat=4.3
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
    +probCat4C+probCat4D)*100:
    tempCat=4.4
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
    +probCat4C+probCat4D+probCat4E)*100:
    tempCat=4.5
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
    +probCat4C+probCat4D+probCat4E+probCat5A)*100:
    tempCat=5.1
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
    +probCat4C+probCat4D+probCat4E+probCat5A+probCat5B)*100:
    tempCat=5.2
elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
    +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
    +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
    +probCat4C+probCat4D+probCat4E+probCat5A+probCat5B+probCat5C)*100:
    tempCat=5.3
```

```
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E
        +probCat2A+probCat2B+probCat2C+probCat2D+probCat2E+probCat3A
        +probCat3B+probCat3C+probCat3D+probCat3E+probCat4A+probCat4B
        +probCat4C+probCat4D+probCat4E+probCat5A+probCat5B+probCat5C
        +probCat5D)*100:
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat

for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')]
                    for line in open(fileName, 'r')]#reading each line of the acces
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
                rows = len(T)
                AccessStartDates=[datetime.datetime.strptime(str(T[i]),
                    '%d %b %Y %H:%M:%S')-datetime.datetime.strptime(dtStart,
                    '%d %b %Y %H:%M:%S') for i in range(rows)]#[datetime.datetime.s
                AccessStartTime=[timedelta.total_seconds
                    (AccessStartDates[i]) for i in range(rows)]
                AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                S=textArray[:,2] #next six lines including this one are the same as
                S=[i.split('.',1)[0] for i in S]
                S=np.array(S)

                AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                ObservationIntervalsTgt=set()
                for i in range(0,rows):
                    for l in range(0,AccessStartCount[i,1]):
                        ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                for i in UBERList[0]:
```

```
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list
    PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1,

#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    singleObservation=UBERList[w][t][i]
                    minilist.append(singleObservation)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    countervalues=[]
                    for x in indices:#This creates a list of the Counter values for the
                        value=UBERList[1][x]
                        countervalues.append(value)
                    maxofmini=max(countervalues)#finds the max Counter value of the vis
                    max_index=countervalues.index(maxofmini)#finds index of max Counter
                    use_this_index=indices[max_index]#finds index of the Counter that c
                    UBERList[1][use_this_index]=0#resets the counter for the selected t
                    usedindices.append(use_this_index)
                    satCount[use_this_index]=satCount[use_this_index]+1
            for g in xrange(0,numTgt):
                if g in (usedindices):
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
else:
    fitness=[1000,(86400/60.0),0]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
```

```
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
    print '4A: ', count4A
    print '4B: ', count4B
    print '4C: ', count4C
    print '4D: ', count4D
    print '4E: ', count4E
    print '5A: ', count5A
    print '5B: ', count5B
    print '5C: ', count5C
    print '5D: ', count5D
    print '5E: ', count5E
    print '1s: ', count1A+count1B+count1C+count1D+count1E
    print '2s: ', count2A+count2B+count2C+count2D+count2E
```

```
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
print len(UBERList[1])
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3.0) and (PriorityList[i]>2.0):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
```

```
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
```

```
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
ones=[]
for i in xrange(0,len(madeThreshold)):
    if PriorityList[i]<2:
        ones.append(madeThreshold[i])
if ones==[]:
    print 'Total Met Target Threshold of Cat 1: N/A'
else:
```

```
        print 'Total Met Target Threshold of Cat 1: ', sum(ones)
    twos=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<3) and (PriorityList[i]>2):
            twos.append(madeThreshold[i])
    if twos==[]:
        print 'Total Met Target Threshold of Cat 2: N/A'
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)

else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
    if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
        fitness[0]=100000
        fitness[1]=100000
        fitness[2]=100000
```

1.1.3 SSN Scheduler Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=190

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
#dtStart = '21 Dec 2018 00:00:00' #Start date & time remember to match these up with th
#dtStop= '22 Dec 2018 00:00:00'  #Stop date & time
#dtStart = '20 Mar 2019 00:00:00' #Start date & time remember to match these up with th
#dtStop= '21 Mar 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
```

```
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])


repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#numTgt=1000
#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
```

```
UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
AllObservationIntervals=[]
for g in xrange(0,Intervals):#Creates the list of time steps
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#PriorityList=[1,2,3,4,5]
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
```

```
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat

for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')] for line in
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
                rows = len(T)
                AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                S=textArray[:,2] #next six lines including this one are the same as
                S=[i.split('.',1)[0] for i in S]
                S=np.array(S)

                AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                ObservationIntervalsTgt=set()
```

```
                    for i in range(0,rows):
                        for l in range(0,AccessStartCount[i,1]):
                            ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                    for i in UBERList[0]:
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list
        #for testing code
    PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1,

#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)
```

```
zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
zenithangleArray=[[str(x) for x in line.strip().split(',')] for lir
zenithangleArray=np.array(zenithangleArray)
LzenithangleList=zenithangleArray[:,1]
TzenithangleList=zenithangleArray[:,2]
LzenithAngles=[]
TzenithAngles=[]
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
```

```
                        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
                        #Sky Background, applicable to ground based telescopes only
                        I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
                        fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
                        Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
                        Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
                        Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                        Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                        BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 w
                        Btot=BZ+Bmoon#Sky Luminance in lamberts
                        Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                        SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                        #print index[y], len(index)
                        #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                        def detect(rRSO):
                            return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                        res=minimize_scalar(detect, method='golden', options={'xtol
                        rRSO=math.fabs(res.x)
                        Dia=2*rRSO #this is the diameter for a single access for a
                        Size.append(Dia)
                        #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                        SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSO
                    except:
                        pass
#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    #print 'UBERList: ', UBERList[3]
    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                snrlist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    if SNRindex[t][i]>=4.0:
                        singleObservation=UBERList[w][t][i]
                        minilist.append(singleObservation)
                    else:
                        minilist.append(0)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    #print indices
                    indices2=[] #this is the matching list of priorities to those in in
                    indicesSatCount=[] #this is a matching list with the count of how m
                    countervalues=[]
                    shortindex=[]
                    TwoEindex=[]
                    TwoEcountervalues=[]
                    one=[]
```

```
oneCat=[]
oneCount=[]
oneQualify=[]
two=[]
twoCat=[]
twoCount=[]
twoQualify=[]
ThreetoFive=[]
ThreetoFiveCat=[]
ThreetoFiveCount=[]
snowyFLAG=0
ThreetoFiveQualify=[]
for j in xrange(0,len(minilist)):
    if minilist[j]==1:
        indices2.append(PriorityList[j])
        indicesSatCount.append(satCount[j])
minofmini2=min(indices2)
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<2:
        one.append(indices[j])
        oneCat.append(indices2[j])
        oneCount.append(indicesSatCount[j])
for j in xrange(0,len(oneCount)): #this section is to make a list c
    if oneCat[j]==1.1:
        if oneCount[j]<limit[0]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.2:
        if oneCount[j]<limit[1]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.3:
        if oneCount[j]<limit[2]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.4:
        if oneCount[j]<limit[3]:
            oneQualify.append(one[j])
    else:
        if oneCount[j]<limit[4]:
            oneQualify.append(one[j])
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<3:
        two.append(indices[j])
        twoCat.append(indices2[j])
        twoCount.append(indicesSatCount[j])
for j in xrange(0,len(twoCount)): #this section is to make a list c
    if twoCat[j]==2.1:
        if twoCount[j]<limit[5]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.2:
        if twoCount[j]<limit[6]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.3:
        if twoCount[j]<limit[7]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.4:
        if twoCount[j]<limit[8]:
```

```
            twoQualify.append(two[j])
    else:
        if twoCount[j]<limit[9]:
            twoQualify.append(two[j])
for j in xrange(0,len(indices)): #this section is to make a list of
    if indices2[j]>=3:
        ThreetoFive.append(indices[j])
        ThreetoFiveCat.append(indices2[j])
        ThreetoFiveCount.append(indicesSatCount[j])
for j in xrange(0,len(ThreetoFiveCount)): #this section is to make
    if ThreetoFiveCat[j]==3.1:
        if ThreetoFiveCount[j]<limit[10]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.2:
        if ThreetoFiveCount[j]<limit[11]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.3:
        if ThreetoFiveCount[j]<limit[12]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.4:
        if ThreetoFiveCount[j]<limit[13]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.5:
        if ThreetoFiveCount[j]<limit[14]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.1:
        if ThreetoFiveCount[j]<limit[15]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.2:
        if ThreetoFiveCount[j]<limit[16]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.3:
        if ThreetoFiveCount[j]<limit[17]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.4:
        if ThreetoFiveCount[j]<limit[18]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.5:
        if ThreetoFiveCount[j]<limit[19]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.1:
        if ThreetoFiveCount[j]<limit[20]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.2:
        if ThreetoFiveCount[j]<limit[21]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.3:
        if ThreetoFiveCount[j]<limit[22]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.4:
        if ThreetoFiveCount[j]<limit[23]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    else:
        if ThreetoFiveCount[j]<limit[24]:
            ThreetoFiveQualify.append(ThreetoFive[j])
```

```
        if ((minofmini2<2) and (oneQualify!=[])):
            for x in oneQualify:
                value=UBERList[1][x]
                countervalues.append(value)
            maxofmini=max(countervalues)#finds the max Counter value of the
            max_index=countervalues.index(maxofmini)#finds index of max Cou
            use_this_index=oneQualify[max_index]#finds index of the Counter
        elif ((minofmini2<2)):
            for j in xrange(0,len(indices)):
                if indices2[j]==minofmini2:
                    shortindex.append(indices[j])
            for x in shortindex:
                value=UBERList[1][x]
                countervalues.append(value)
            maxofmini=max(countervalues)#finds the max Counter value of the
            max_index=countervalues.index(maxofmini)#finds index of max Cou
            use_this_index=shortindex[max_index]#finds index of the Counter
        elif ((minofmini2<3) and (twoQualify!=[])):
            for x in twoQualify:
                value=UBERList[1][x]
                countervalues.append(value)
            maxofmini=max(countervalues)#finds the max Counter value of the
            max_index=countervalues.index(maxofmini)#finds index of max Cou
            use_this_index=twoQualify[max_index]#finds index of the Counter
        elif ((minofmini2<3)):
            for j in xrange(0,len(indices)):
                if indices2[j]==minofmini2:
                    shortindex.append(indices[j])
            for x in shortindex:
                value=UBERList[1][x]
                countervalues.append(value)
            maxofmini=max(countervalues)#finds the max Counter value of the
            max_index=countervalues.index(maxofmini)#finds index of max Cou
            use_this_index=shortindex[max_index]#finds index of the Counter
        elif ((minofmini2>=3) and (ThreetoFiveQualify!=[])):
            for x in ThreetoFiveQualify:
                value=UBERList[1][x]
                countervalues.append(value)
            maxofmini=max(countervalues)#finds the max Counter value of the
            max_index=countervalues.index(maxofmini)#finds index of max Cou
            use_this_index=ThreetoFiveQualify[max_index]#finds index of the
        else: #if not cat 1.1-2.5 and snowy tables are maxed, treat ALL sat
            for x in indices:#This creates a list of the Counter values for
                value=UBERList[1][x]
                countervalues.append(value)
            maxofmini=max(countervalues)#finds the max Counter value of the
            max_index=countervalues.index(maxofmini)#finds index of max Cou
            use_this_index=indices[max_index]#finds index of the Counter th
        #print use_this_index
        UBERList[1][use_this_index]=0#resets the counter for the selected t
        usedindices.append(use_this_index)
        satCount[use_this_index]=satCount[use_this_index]+1
        #print SNRindex[use_this_index][i]
for g in xrange(0,numTgt):
    if g in (usedindices):
```

```
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi
    #%%
    stopT=time.time()
    runTime=(stopT-staT)
else:
    fitness=[(86400/60.0)]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
```

```
print '2C: ', count2C
print '2D: ', count2D
print '2E: ', count2E
print '3A: ', count3A
print '3B: ', count3B
print '3C: ', count3C
print '3D: ', count3D
print '3E: ', count3E
print '4A: ', count4A
print '4B: ', count4B
print '4C: ', count4C
print '4D: ', count4D
print '4E: ', count4E
print '5A: ', count5A
print '5B: ', count5B
print '5C: ', count5C
print '5D: ', count5D
print '5E: ', count5E
print '1s: ', count1A+count1B+count1C+count1D+count1E
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
```

```
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
```

```
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
```

```
            threshold.append(limit[22])
        elif PriorityList[i]==5.4:
            threshold.append(limit[23])
        else:
            threshold.append(limit[24])
    madeThreshold=[]
    for i in xrange(0,len(satCount)):
        if satCount[i]>=threshold[i]:
            madeThreshold.append(1)
        else:
            madeThreshold.append(0)
    print 'Total Met Target Threshold All: ', sum(madeThreshold)
    ones=[]
    for i in xrange(0,len(madeThreshold)):
        if PriorityList[i]<2:
            ones.append(madeThreshold[i])
    if ones==[]:
        print 'Total Met Target Threshold of Cat 1: N/A'
    else:
        print 'Total Met Target Threshold of Cat 1: ', sum(ones)
    twos=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<3) and (PriorityList[i]>2):
            twos.append(madeThreshold[i])
    if twos==[]:
        print 'Total Met Target Threshold of Cat 2: N/A'
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)
    #print UBERList[1]

else:
```

```
workPath = os.environ['LOC']
workSpace = os.environ['WORKDIR']
repPath = os.path.join(workSpace,'Reports_Jan')
scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
os.chdir(scorePath)
fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
#below are the penalty parameters and the gradient of the second tier of the penaly
valMaxSize=75
valMaxLat=90
valMaxCost=30
gradSize=valMaxSize*1.1
gradLat=valMaxLat*1.1
gradCost=valMaxCost*1.1
#Here is where the penalty comes in, after the score is computed then it is accesse
if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
    fitness[0]=100000
    fitness[1]=100000
    fitness[2]=100000
```

1.1.4 Relaxed SSN Scheduler Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
#import pty
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=190

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
```

1

```
probCat4E=0.05
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
#dtStart = '21 Dec 2018 00:00:00' #Start date & time remember to match these up with th
#dtStop= '22 Dec 2018 00:00:00'  #Stop date & time
#dtStart = '20 Mar 2019 00:00:00' #Start date & time remember to match these up with th
#dtStop= '21 Mar 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
```

```
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])

repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#numTgt=600
#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
```

```
UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
AllObservationIntervals=[]
for g in xrange(0,Intervals):#Creates the list of time steps
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#PriorityList=[1,2,3,4,5]
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
```

```
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat

for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')] for line in
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
                rows = len(T)
                AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                S=textArray[:,2] #next six lines including this one are the same as
                S=[i.split('.',1)[0] for i in S]
                S=np.array(S)

                AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                ObservationIntervalsTgt=set()
```

```
                    for i in range(0,rows):
                        for l in range(0,AccessStartCount[i,1]):
                            ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                    for i in UBERList[0]:
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list

        #for testing code
    PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1,
#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)
```

```
zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
zenithangleArray=[[str(x) for x in line.strip().split(',')] for lir
zenithangleArray=np.array(zenithangleArray)
LzenithangleList=zenithangleArray[:,1]
TzenithangleList=zenithangleArray[:,2]
LzenithAngles=[]
TzenithAngles=[]
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
```

```
                        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
                        #Sky Background, applicable to ground based telescopes only
                        I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
                        fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
                        Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
                        Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
                        Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                        Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                        BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 w
                        Btot=BZ+Bmoon#Sky luminance in lamberts
                        Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                        SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                        #print index[y], len(index)
                        #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                        def detect(rRSO):
                            return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                        res=minimize_scalar(detect, method='golden', options={'xtol
                        rRSO=math.fabs(res.x)
                        Dia=2*rRSO #this is the diameter for a single access for a
                        Size.append(Dia)
                        #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                        SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSO
                    except:
                        pass
#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                snrlist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    if SNRindex[t][i]>=4.0:
                        singleObservation=UBERList[w][t][i]
                        minilist.append(singleObservation)
                    else:
                        minilist.append(0)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    indices2=[] #this is the matching list of priorities to those in in
                    indicesSatCount=[] #this is a matching list with the count of how m
                    countervalues=[]
                    shortindex=[]
                    TwoEindex=[]
                    TwoEcountervalues=[]
                    one=[]
                    oneCat=[]
                    oneCount=[]
```

```
oneQualify=[]
two=[]
twoCat=[]
twoCount=[]
twoQualify=[]
ThreetoFive=[]
ThreetoFiveCat=[]
ThreetoFiveCount=[]
ThreetoFiveQualify=[]
for j in xrange(0,len(minilist)):
    if minilist[j]==1:
        indices2.append(PriorityList[j])
        indicesSatCount.append(satCount[j])
minofmini2=min(indices2)
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<2:
        one.append(indices[j])
        oneCat.append(indices2[j])
        oneCount.append(indicesSatCount[j])
for j in xrange(0,len(oneCount)): #this section is to make a list c
    if oneCat[j]==1.1:
        if oneCount[j]<limit[0]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.2:
        if oneCount[j]<limit[1]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.3:
        if oneCount[j]<limit[2]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.4:
        if oneCount[j]<limit[3]:
            oneQualify.append(one[j])
    else:
        if oneCount[j]<limit[4]:
            oneQualify.append(one[j])
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<3:
        two.append(indices[j])
        twoCat.append(indices2[j])
        twoCount.append(indicesSatCount[j])
for j in xrange(0,len(twoCount)): #this section is to make a list c
    if twoCat[j]==2.1:
        if twoCount[j]<limit[5]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.2:
        if twoCount[j]<limit[6]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.3:
        if twoCount[j]<limit[7]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.4:
        if twoCount[j]<limit[8]:
            twoQualify.append(two[j])
    else:
        if twoCount[j]<limit[9]:
```

```
            twoQualify.append(two[j])
for j in xrange(0,len(indices)): #this section is to make a list of
    if indices2[j]>=3:
        ThreetoFive.append(indices[j])
        ThreetoFiveCat.append(indices2[j])
        ThreetoFiveCount.append(indicesSatCount[j])
for j in xrange(0,len(ThreetoFiveCount)): #this section is to make
    if ThreetoFiveCat[j]==3.1:
        if ThreetoFiveCount[j]<limit[10]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.2:
        if ThreetoFiveCount[j]<limit[11]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.3:
        if ThreetoFiveCount[j]<limit[12]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.4:
        if ThreetoFiveCount[j]<limit[13]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.5:
        if ThreetoFiveCount[j]<limit[14]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.1:
        if ThreetoFiveCount[j]<limit[15]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.2:
        if ThreetoFiveCount[j]<limit[16]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.3:
        if ThreetoFiveCount[j]<limit[17]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.4:
        if ThreetoFiveCount[j]<limit[18]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.5:
        if ThreetoFiveCount[j]<limit[19]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.1:
        if ThreetoFiveCount[j]<limit[20]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.2:
        if ThreetoFiveCount[j]<limit[21]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.3:
        if ThreetoFiveCount[j]<limit[22]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.4:
        if ThreetoFiveCount[j]<limit[23]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    else:
        if ThreetoFiveCount[j]<limit[24]:
            ThreetoFiveQualify.append(ThreetoFive[j])
if ((minofmini2<2) and (oneQualify!=[])):
    for x in oneQualify:
        value=UBERList[1][x]
```

```
                        countervalues.append(value)
                    maxofmini=max(countervalues)#finds the max Counter value of the
                    max_index=countervalues.index(maxofmini)#finds index of max Cou
                    use_this_index=oneQualify[max_index]#finds index of the Counter
                elif ((minofmini2<3) and (twoQualify!=[])):
                    for x in twoQualify:
                        value=UBERList[1][x]
                        countervalues.append(value)
                    maxofmini=max(countervalues)#finds the max Counter value of the
                    max_index=countervalues.index(maxofmini)#finds index of max Cou
                    use_this_index=twoQualify[max_index]#finds index of the Counter
                elif ((minofmini2>=3) and (ThreetoFiveQualify!=[])):
                    for x in ThreetoFiveQualify:
                        value=UBERList[1][x]
                        countervalues.append(value)
                    maxofmini=max(countervalues)#finds the max Counter value of the
                    max_index=countervalues.index(maxofmini)#finds index of max Cou
                    use_this_index=ThreetoFiveQualify[max_index]#finds index of the
                else: #if not cat 1.1-2.5 and snowy tables are maxed, treat ALL sat
                    for x in indices:#This creates a list of the Counter values for
                        value=UBERList[1][x]
                        countervalues.append(value)
                    maxofmini=max(countervalues)#finds the max Counter value of the
                    max_index=countervalues.index(maxofmini)#finds index of max Cou
                    use_this_index=indices[max_index]#finds index of the Counter th
                UBERList[1][use_this_index]=0#resets the counter for the selected t
                usedindices.append(use_this_index)
                satCount[use_this_index]=satCount[use_this_index]+1
            for g in xrange(0,numTgt):
                if g in (usedindices):
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
```

```
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
    print '4A: ', count4A
    print '4B: ', count4B
    print '4C: ', count4C
    print '4D: ', count4D
    print '4E: ', count4E
    print '5A: ', count5A
    print '5B: ', count5B
    print '5C: ', count5C
    print '5D: ', count5D
    print '5E: ', count5E
    print '1s: ', count1A+count1B+count1C+count1D+count1E
    print '2s: ', count2A+count2B+count2C+count2D+count2E
    print '3s: ', count3A+count3B+count3C+count3D+count3E
    print '4s: ', count4A+count4B+count4C+count4D+count4E
```

```
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
```

```
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
```

```
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
ones=[]
for i in xrange(0,len(madeThreshold)):
    if PriorityList[i]<2:
        ones.append(madeThreshold[i])
if ones==[]:
    print 'Total Met Target Threshold of Cat 1: N/A'
else:
    print 'Total Met Target Threshold of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(madeThreshold)):
```

```
            if (PriorityList[i]<3) and (PriorityList[i]>2):
                twos.append(madeThreshold[i])
        if twos==[]:
            print 'Total Met Target Threshold of Cat 2: N/A'
        else:
            print 'Total Met Target Threshold of Cat 2: ', sum(twos)
        threes=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]<4) and (PriorityList[i]>3):
                threes.append(madeThreshold[i])
        if threes==[]:
            print 'Total Met Target Threshold of Cat 3: N/A'
        else:
            print 'Total Met Target Threshold of Cat 3: ', sum(threes)
        fours=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]<5) and (PriorityList[i]>4):
                fours.append(madeThreshold[i])
        if fours==[]:
            print 'Total Met Target Threshold of Cat 4: N/A'
        else:
            print 'Total Met Target Threshold of Cat 4: ', sum(fours)
        fives=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]>5):
                fives.append(madeThreshold[i])
        if fives==[]:
            print 'Total Met Target Threshold of Cat 5: N/A'
        else:
            print 'Total Met Target Threshold of Cat 5: ', sum(fives)
    else:
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(scorePath)
        fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
        #below are the penalty parameters and the gradient of the second tier of the penaly
        valMaxSize=75
        valMaxLat=90
        valMaxCost=30
        gradSize=valMaxSize*1.1
        gradLat=valMaxLat*1.1
        gradCost=valMaxCost*1.1
        #Here is where the penalty comes in, after the score is computed then it is accesse
        if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
            fitness[0]=100000
            fitness[1]=100000
            fitness[2]=100000
```

1.1.5 Relaxed SSN Scheduler Model with Spacing

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
#import pty
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30

numTgt=190

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
```

```
probCat4D=0.05
probCat4E=0.05
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1
#probCat1=0.11
#probCat2A=0.11
#probCat2B=0.11
#probCat2C=0.11
#probCat2D=0.11
#probCat2E=0.11
#probCat3=0.11
#probCat4=0.11
#probCat5=0.12

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

#number of intervals between NEEDing to look at it (if I just looked at it not too long
spread=60

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
#dtStart = '21 Dec 2018 00:00:00' #Start date & time remember to match these up with th
#dtStop= '22 Dec 2018 00:00:00'  #Stop date & time
#dtStart = '20 Mar 2019 00:00:00' #Start date & time remember to match these up with th
#dtStop= '21 Mar 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
```

```
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])


repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)



#numTgt=600
#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
```

```
PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Co
spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
    #blackbody (approximation for sun)
QE=0.65 #Quantum efficiency
opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
SNR=6 #Minimum signal to noise ratio permitting detection
refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
    #400nm to 800nm (min to max nm)
numObservers=3
UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
AllObservationIntervals=[]
for g in xrange(0,Intervals):#Creates the list of time steps
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#PriorityList=[1,2,3,4,5]
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
```

```
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat

for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')] for line in
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
```

```
                    rows = len(T)
                    AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                    AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                    AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                    S=textArray[:,2] #next six lines including this one are the same as
                    S=[i.split('.',1)[0] for i in S]
                    S=np.array(S)

                    AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                    AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                    AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                    IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                    IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                    AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                    AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                    ObservationIntervalsTgt=set()
                    for i in range(0,rows):
                        for l in range(0,AccessStartCount[i,1]):
                            ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                    for i in UBERList[0]:
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ec
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list

        #for testing code
    PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1,

#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
```

```
if flag==1:
    gndrangeArray=np.array(gndrangeArray)
    gndT=gndrangeArray[:,0]
    gndT=[lin.split('.',1)[0] for lin in gndT]
    gndT=np.array(gndT)
    rows = len(gndT)
    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
    #print ObsStartTime
    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
    R=gndrangeArray[:,3]
    R=[int(ran.split('.',1)[0]) for ran in R]
    # This section determines the indices of ObsStartIntervals (which c
    Ranges=[]
    for k in index:
        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
        Range=R[Rindex]
        Ranges.append(Range)

    zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
    zenithangleArray=[[str(x) for x in line.strip().split(',')] for lin
    zenithangleArray=np.array(zenithangleArray)
    LzenithangleList=zenithangleArray[:,1]
    TzenithangleList=zenithangleArray[:,2]
    LzenithAngles=[]
    TzenithAngles=[]
    for k in index:
        try:
            LzA=float(LzenithangleList[k])
            TzA=float(TzenithangleList[k])
            LzenithAngles.append(LzA)
            TzenithAngles.append(TzA)
        except:
            pass
    anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
    angleArray=[[str(x) for x in line.strip().split(',')] for line in c
    angleArray=np.array(angleArray)
    phaseangleList=angleArray[:,1]
    lunarphaseangleList=angleArray[:,2]
    PhaseAngles=[]
    lunarPhaseAngles=[]
    for k in index:
        try:
            PhsA=float(phaseangleList[k])
            LPhsA=float(lunarphaseangleList[k])
            PhaseAngles.append(PhsA)
            lunarPhaseAngles.append(LPhsA)
        except:
            pass
    #print len(index)
    for y in range(0, len(index)):
        #print y
        try:
            Tint=1.0
            Range=Ranges[y]*1000# this will come from STK
            phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
```

```
                        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
                        lunarzenith=LzenithAngles[y] #(degrees) this value will com
                        targetzenith=TzenithAngles[y]
                        #Variable from design scripted in python
                        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
                        d=D*0.3 #obscuration diameter, variable from system design
                        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
                        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astro
                        #General Calcs
                        focalLength=2*D #Assumes a fast, but not quite state of the
                        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
                        pixPitch=focalLength*IFOV
                        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
                        #it is motion of the GEO belt relative to the star backgrou
                        #(worstcase when the image is between two pixel rows)
                        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
                        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
                        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
                        #Sky Background, applicable to ground based telescopes only
                        I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
                        fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
                        Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
                        Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
                        Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                        Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                        BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 w
                        Btot=BZ+Bmoon#Sky luminance in lamberts
                        Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                        SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                        #print index[y], len(index)
                        #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                        def detect(rRSO):
                            return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                        res=minimize_scalar(detect, method='golden', options={'xtol
                        rRSO=math.fabs(res.x)
                        Dia=2*rRSO #this is the diameter for a single access for a
                        Size.append(Dia)
                        #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                        SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSC
                    except:
                        pass
#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                snrlist=[]
```

```
for t in xrange(0,numTgt):#this section creates a "minilist" that is a
    if SNRindex[t][i]>=4.0:
        singleObservation=UBERList[w][t][i]
        minilist.append(singleObservation)
    else:
        minilist.append(0)
if sum(minilist)!=0:
    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
    indices2=[] #this is the matching list of priorities to those in in
    indicesSatCount=[] #this is a matching list with the count of how m
    SPREADindices=[]
    SPREADindices2=[]
    SPREADindicesSatCount=[]
    countervalues=[]
    SPREADcountervalues=[]
    shortindex=[]
    TwoEindex=[]
    TwoEcountervalues=[]
    one=[]
    oneCat=[]
    oneCount=[]
    oneQualify=[]
    two=[]
    twoCat=[]
    twoCount=[]
    twoQualify=[]
    ThreetoFive=[]
    ThreetoFiveCat=[]
    ThreetoFiveCount=[]
    ThreetoFiveQualify=[]
    for j in xrange(0,len(minilist)):
        if minilist[j]==1:
            indices2.append(PriorityList[j])
            indicesSatCount.append(satCount[j])
    minofmini2=min(indices2)
    for x in indices:#This section is creating a list of those visible
        value=UBERList[1][x]
        SPREADcountervalues.append(value)
    for j in xrange(0,len(SPREADcountervalues)):
        if SPREADcountervalues[j]>spread:
            SPREADindices.append(indices[j])
            SPREADindices2.append(indices2[j])
            SPREADindicesSatCount.append(indicesSatCount[j])
    for j in xrange(0,len(SPREADindices)): #this section is to make lis
        if SPREADindices2[j]<2:
            one.append(SPREADindices[j])
            oneCat.append(SPREADindices2[j])
            oneCount.append(SPREADindicesSatCount[j])
    for j in xrange(0,len(oneCount)): #this section is to make a list c
        if oneCat[j]==1.1:
            if oneCount[j]<limit[0]:
                oneQualify.append(one[j])
        elif oneCat[j]==1.2:
            if oneCount[j]<limit[1]:
                oneQualify.append(one[j])
```

```
    elif oneCat[j]==1.3:
        if oneCount[j]<limit[2]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.4:
        if oneCount[j]<limit[3]:
            oneQualify.append(one[j])
    else:
        if oneCount[j]<limit[4]:
            oneQualify.append(one[j])
for j in xrange(0,len(SPREADindices)): #this section is to make lis
    if SPREADindices2[j]<3:
        two.append(SPREADindices[j])
        twoCat.append(SPREADindices2[j])
        twoCount.append(SPREADindicesSatCount[j])
for j in xrange(0,len(twoCount)): #this section is to make a list c
    if twoCat[j]==2.1:
        if twoCount[j]<limit[5]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.2:
        if twoCount[j]<limit[6]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.3:
        if twoCount[j]<limit[7]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.4:
        if twoCount[j]<limit[8]:
            twoQualify.append(two[j])
    else:
        if twoCount[j]<limit[9]:
            twoQualify.append(two[j])
for j in xrange(0,len(SPREADindices)): #this section is to make a l
    if SPREADindices2[j]>=3:
        ThreetoFive.append(SPREADindices[j])
        ThreetoFiveCat.append(SPREADindices2[j])
        ThreetoFiveCount.append(SPREADindicesSatCount[j])
for j in xrange(0,len(ThreetoFiveCount)): #this section is to make
    if ThreetoFiveCat[j]==3.1:
        if ThreetoFiveCount[j]<limit[10]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.2:
        if ThreetoFiveCount[j]<limit[11]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.3:
        if ThreetoFiveCount[j]<limit[12]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.4:
        if ThreetoFiveCount[j]<limit[13]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.5:
        if ThreetoFiveCount[j]<limit[14]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.1:
        if ThreetoFiveCount[j]<limit[15]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.2:
```

```
            if ThreetoFiveCount[j]<limit[16]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.3:
            if ThreetoFiveCount[j]<limit[17]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.4:
            if ThreetoFiveCount[j]<limit[18]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.5:
            if ThreetoFiveCount[j]<limit[19]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.1:
            if ThreetoFiveCount[j]<limit[20]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.2:
            if ThreetoFiveCount[j]<limit[21]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.3:
            if ThreetoFiveCount[j]<limit[22]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.4:
            if ThreetoFiveCount[j]<limit[23]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        else:
            if ThreetoFiveCount[j]<limit[24]:
                ThreetoFiveQualify.append(ThreetoFive[j])
    if ((minofmini2<2) and (oneQualify!=[])):
        for x in oneQualify:
            value=UBERList[1][x]
            countervalues.append(value)
        maxofmini=max(countervalues)#finds the max Counter value of the
        max_index=countervalues.index(maxofmini)#finds index of max Cou
        use_this_index=oneQualify[max_index]#finds index of the Counter
    elif ((minofmini2<3) and (twoQualify!=[])):
        for x in twoQualify:
            value=UBERList[1][x]
            countervalues.append(value)
        maxofmini=max(countervalues)#finds the max Counter value of the
        max_index=countervalues.index(maxofmini)#finds index of max Cou
        use_this_index=twoQualify[max_index]#finds index of the Counter
    elif ((minofmini2>=3) and (ThreetoFiveQualify!=[])):
        for x in ThreetoFiveQualify:
            value=UBERList[1][x]
            countervalues.append(value)
        maxofmini=max(countervalues)#finds the max Counter value of the
        max_index=countervalues.index(maxofmini)#finds index of max Cou
        use_this_index=ThreetoFiveQualify[max_index]#finds index of the
    else: #if not cat 1.1-2.5 and snowy tables are maxed, treat ALL sat
        for x in indices:#This creates a list of the Counter values for
            value=UBERList[1][x]
            countervalues.append(value)
        maxofmini=max(countervalues)#finds the max Counter value of the
        max_index=countervalues.index(maxofmini)#finds index of max Cou
        use_this_index=indices[max_index]#finds index of the Counter th
    UBERList[1][use_this_index]=0#resets the counter for the selected t
```

```
                    #print UBERList[1]
                    usedindices.append(use_this_index)
                    satCount[use_this_index]=satCount[use_this_index]+1
            for g in xrange(0,numTgt):
                if g in (usedindices):
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi
    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
```

```
print '1A: ', count1A
print '1B: ', count1B
print '1C: ', count1C
print '1D: ', count1D
print '1E: ', count1E
print '2A: ', count2A
print '2B: ', count2B
print '2C: ', count2C
print '2D: ', count2D
print '2E: ', count2E
print '3A: ', count3A
print '3B: ', count3B
print '3C: ', count3C
print '3D: ', count3D
print '3E: ', count3E
print '4A: ', count4A
print '4B: ', count4B
print '4C: ', count4C
print '4D: ', count4D
print '4E: ', count4E
print '5A: ', count5A
print '5B: ', count5B
print '5C: ', count5C
print '5D: ', count5D
print '5E: ', count5E
print '1s: ', count1A+count1B+count1C+count1D+count1E
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
```

```
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
```

```
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
```

```
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
ones=[]
for i in xrange(0,len(madeThreshold)):
    if PriorityList[i]<2:
        ones.append(madeThreshold[i])
if ones==[]:
    print 'Total Met Target Threshold of Cat 1: N/A'
else:
    print 'Total Met Target Threshold of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(madeThreshold)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(madeThreshold[i])
if twos==[]:
    print 'Total Met Target Threshold of Cat 2: N/A'
else:
    print 'Total Met Target Threshold of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(madeThreshold)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(madeThreshold[i])
if threes==[]:
    print 'Total Met Target Threshold of Cat 3: N/A'
else:
    print 'Total Met Target Threshold of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(madeThreshold)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(madeThreshold[i])
if fours==[]:
    print 'Total Met Target Threshold of Cat 4: N/A'
else:
    print 'Total Met Target Threshold of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(madeThreshold)):
    if (PriorityList[i]>5):
        fives.append(madeThreshold[i])
```

```
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)
else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
    if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
        fitness[0]=100000
        fitness[1]=100000
        fitness[2]=100000
```

1.1.6 Integer Program Setup

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will setup the data required for IP schedulers.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
#import pty
import commands
from clearSky import clearSkySetList
import random
from pulp import *

TS=86400
repTS=30
numTgt=190

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

1

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
#dtStart = '21 Dec 2018 00:00:00' #Start date & time remember to match these up with th
#dtStop= '22 Dec 2018 00:00:00'  #Stop date & time
#dtStart = '20 Mar 2019 00:00:00' #Start date & time remember to match these up with th
#dtStop= '21 Mar 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
```

```
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])


repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20070
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
```

```
AllObservationIntervals=[]
for g in xrange(0,Intervals):#Creates the list of time steps
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#PriorityList=[1,2,3,4,5]
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    if random100<=probCat1A*100:
        tempCat=1.1
    elif random100<=(probCat1A+probCat1B)*100:
        tempCat=1.2
    elif random100<=(probCat1A+probCat1B+probCat1C)*100:
        tempCat=1.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D)*100:
        tempCat=1.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
```

```
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat


for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')] for line in
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
                rows = len(T)
                AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                S=textArray[:,2] #next six lines including this one are the same as
                S=[i.split('.',1)[0] for i in S]
                S=np.array(S)

                AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                ObservationIntervalsTgt=set()
                for i in range(0,rows):
                    for l in range(0,AccessStartCount[i,1]):
```

```
                        ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                for i in UBERList[0]:
                    if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                        BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list

        #for testing code
    PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1,

#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)
```

```
zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
zenithangleArray=[[str(x) for x in line.strip().split(',')] for lir
zenithangleArray=np.array(zenithangleArray)
LzenithangleList=zenithangleArray[:,1]
TzenithangleList=zenithangleArray[:,2]
LzenithAngles=[]
TzenithAngles=[]
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
```

```
                                #Sky Background, applicable to ground based telescopes only
                                I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
                                fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
                                Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
                                Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
                                Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                                Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                                BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 k
                                Btot=BZ+Bmoon#Sky luminance in lamberts
                                Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                                SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                                #print index[y], len(index)
                                #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                                def detect(rRSO):
                                    return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                                    #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                                res=minimize_scalar(detect, method='golden', options={'xtol
                                rRSO=math.fabs(res.x)
                                Dia=2*rRSO #this is the diameter for a single access for a
                                Size.append(Dia)
                                #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                                SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSC
                            except:
                                pass

    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    givenAvailability=[]
    for x in xrange(0,len(UBERList[3])):
        temp=[]
        for y in xrange(0,len(UBERList[3][0])):
            if (UBERList[3][x][y]==1) and SNRindex[x][y]>=6.0:
                temp.append(1)
            else:
                temp.append(0)
        givenAvailability.append(temp)

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]
model_lp_out = open('model_lp_out.txt','w')


# The status of the solution is printed to the screen
print "\n","Status:"
model_lp_out.write("ANYTHING")
model_lp_out.write("]")
model_lp_out.write("\n+-------+-------+-------+\n\n")
print "Objective Value = "
print "m Value = "
model_lp_out.close()
```

```
countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)

else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
    if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
        fitness[0]=100000
        fitness[1]=100000
        fitness[2]=100000
```

1.1.7 Integer Program Evaluation

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will evaluate the IP schedule completed in previous scripts.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
#import pty
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=190

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
probCat5A=0.1
```

1

```
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
#dtStart = '21 Dec 2018 00:00:00' #Start date & time remember to match these up with th
#dtStop= '22 Dec 2018 00:00:00'  #Stop date & time
#dtStart = '20 Mar 2019 00:00:00' #Start date & time remember to match these up with th
#dtStop= '21 Mar 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
```

```
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])


repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)



#numTgt=1000
#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

#         #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
```

```
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
    for g in xrange(0,Intervals):#Creates the list of time steps
        interval=0+g
        AllObservationIntervals.append(interval)
    UBERList.append(AllObservationIntervals)
    Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
    UBERList.append(Counter)
    IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
    UBERList.append(IDCounter)
    #PriorityList=[0]*numTgt #creates and initializes a list of priorities
    satCount=[0]*numTgt #this creates and initializes the list which holds how many tim

    PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1,

#     print "Solutions Written to model_lp_out.txt\n"
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3
    LPSolution=solution

    for i in xrange(0,Intervals): #travel down the intervals
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        for j in xrange(0,numTgt): #travel down the objects
            if LPSolution[j][i] == 1:
                UBERList[1][j] = 0 #resets the counter
                satCount[j] = satCount[j] + 1
#code for pareto
    penalty_neg=[5, 5, 3, 0, 0, 0, 0, 1, 3, 1, 50, 1, 3, 50, 3, 5, 5, 1, 0, 3, 3, 1, 1,
    penalty_pos=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    resetTime=[0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0,
    leftside=0
    rightside=0
    for j in xrange(0,numTgt): #travel down the intervals
        temp=0
        for i in xrange(0,Intervals): #travel down the objects
            temp += w1*solution[j][i]*weights[j][0]
        leftside += temp-w1*penalty_pos[j]*weights[j][1]-w1*penalty_neg[j]*weights[j][2
        rightside += w2*AgeWeights[j]*resetTime[j]
    print 'leftside = ', leftside/w1
    print 'rightside = ', rightside/w2

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
```

```
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]


plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
    print '4A: ', count4A
    print '4B: ', count4B
    print '4C: ', count4C
    print '4D: ', count4D
```

```
print '4E: ', count4E
print '5A: ', count5A
print '5B: ', count5B
print '5C: ', count5C
print '5D: ', count5D
print '5E: ', count5E
print '1s: ', count1A+count1B+count1C+count1D+count1E
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
```

```
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
```

```
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
```

```
    ones=[]
    for i in xrange(0,len(madeThreshold)):
        if PriorityList[i]<2:
            ones.append(madeThreshold[i])
    if ones==[]:
        print 'Total Met Target Threshold of Cat 1: N/A'
    else:
        print 'Total Met Target Threshold of Cat 1: ', sum(ones)
    twos=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<3) and (PriorityList[i]>2):
            twos.append(madeThreshold[i])
    if twos==[]:
        print 'Total Met Target Threshold of Cat 2: N/A'
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)
else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
```

```
if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
    fitness[0]=100000
    fitness[1]=100000
    fitness[2]=100000
```

1.1.8 Binary Integer Program Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

import time
import platform

PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1, 5.4

numTgt=190
UBERList = [[],[],[]]
UBERList.append(givenAvailability)

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]
m=2880 #Intervals

staT=time.time()
plat=platform.system()
    #%%
solution=[]
pos_solution=[]
neg_solution=[]
# Import PuLP modeler functions
from pulp import *
given = UBERList[3]#[[1,1,1,1,1],[0,1,1,0,0],[0,0,1,1,0]]
lpCols = len(given[0])
lpRows = len(given)
Rows = []
Cols = []
Track=[]
for i in xrange(1,lpCols+1):
    Cols.append(i)
for i in xrange(1,lpRows+1):
    Rows.append(i)
for i in xrange(0,lpRows):
    if PriorityList[i]==1.1:
        temp=limit[0]
    elif PriorityList[i]==1.2:
        temp=limit[1]
    elif PriorityList[i]==1.3:
        temp=limit[2]
    elif PriorityList[i]==1.4:
        temp=limit[3]
    elif PriorityList[i]==1.5:
        temp=limit[4]
    elif PriorityList[i]==2.1:
        temp=limit[5]
```

1

```
    elif PriorityList[i]==2.2:
        temp=limit[6]
    elif PriorityList[i]==2.3:
        temp=limit[7]
    elif PriorityList[i]==2.4:
        temp=limit[8]
    elif PriorityList[i]==2.5:
        temp=limit[9]
    elif PriorityList[i]==3.1:
        temp=limit[10]
    elif PriorityList[i]==3.2:
        temp=limit[11]
    elif PriorityList[i]==3.3:
        temp=limit[12]
    elif PriorityList[i]==3.4:
        temp=limit[13]
    elif PriorityList[i]==3.5:
        temp=limit[14]
    elif PriorityList[i]==4.1:
        temp=limit[15]
    elif PriorityList[i]==4.2:
        temp=limit[16]
    elif PriorityList[i]==4.3:
        temp=limit[17]
    elif PriorityList[i]==4.4:
        temp=limit[18]
    elif PriorityList[i]==4.5:
        temp=limit[19]
    elif PriorityList[i]==5.1:
        temp=limit[20]
    elif PriorityList[i]==5.2:
        temp=limit[21]
    elif PriorityList[i]==5.3:
        temp=limit[22]
    elif PriorityList[i]==5.4:
        temp=limit[23]
    else:
        temp=limit[24]
    Track.append(temp)

# The prob variable is created to contain the problem data
prob = LpProblem("IP Problem",LpMaximize)

# The problem variables are created
choices = LpVariable.dicts("Choice",(Rows,Cols),0,1,LpInteger)
penalty_pos = LpVariable.dicts("Over",(Rows),0,m,LpInteger)
penalty_neg = LpVariable.dicts("Under",(Rows),0,m,LpInteger)

weights=[]
for i in xrange(0,lpRows):
    if PriorityList[i]<2:
        weights.append([1,0.6,0.5])
    elif PriorityList[i]<3:
        weights.append([1,0.7,0.4])
    elif PriorityList[i]<4:
```

```
        weights.append([1,0.8,0.3])
    elif PriorityList[i]<5:
        weights.append([1,0.9,0.2])
    else:
        weights.append([1,0.95,0.1])

# The objective function is added
prob += lpSum((choices[r][c]*weights[r-1][0] for c in Cols)-penalty_pos[r]
    *weights[r-1][1]-penalty_neg[r]*weights[r-1][2] for r in Rows),
     "Objective Function"

for c in Cols:
    prob += lpSum(choices[r][c] for r in Rows) <= 3,""

for r in Rows:
    prob += lpSum((choices[r][c] for c in Cols)-penalty_pos[r]
        +penalty_neg[r]) == Track[r-1],""

# The starting numbers are entered as constraints
for x in xrange(0,lpRows):
    for y in xrange(0,lpCols):
        if given[x][y] == 0:
            prob += choices[x+1][y+1] == 0,""

# The problem data is written to an .lp file
prob.writeLP("model_lp_out.lp")

# A file called model_lp_out.txt is created/overwritten for writing to
model_lp_out = open('model_lp_out.txt','w')
solCount = 0
#while True:
prob.solve()

# The status of the solution is printed to the screen
print "\n","Status:", LpStatus[prob.status]
# The solution is printed if it was deemed "optimal" i.e met the constraints
if LpStatus[prob.status] == "Optimal":
    solCount += 1
    # The solution is written to the model_lp_out.txt file
    model_lp_out.write("[")
    for r in Rows:
        model_lp_out.write("[")
        pseudoSol=[]
        for c in Cols:
                pseudoSol.append(int(value(choices[r][c])))
                vee = str(int(value(choices[r][c])))
                model_lp_out.write(vee)
                if (c != lpCols):
                    model_lp_out.write(", ")
                if c == lpCols:
                    model_lp_out.write("]")
                if (c == lpCols) and (r != lpRows):
                    model_lp_out.write(", ")
        solution.append(pseudoSol)
        pos_solution.append(penalty_pos[r])
```

```
        neg_solution.append(penalty_neg[r])
    model_lp_out.write("]")
    model_lp_out.write("\n+-------+-------+-------+\n\n")

    # Each of the variables is printed with it's resolved optimum value
#    for v in prob.variables():
#        print v.name, "=", v.varValue
    print "Objective Value = ", value(prob.objective)
    print "m Value = ", len(UBERList[3])
model_lp_out.close()
# The location of the solutions is give to the user
print "Solutions Written to model_lp_out.txt\n"
#print solution
stopT=time.time()
runTime=(stopT-staT)
print 'Runtime: ' +str(runTime)
```

1.1.9 Multi-Objective Binary Integer Program Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""
import time
import platform

#Given Info
PriorityList=[5.3, 4.3, 5.4, 4.1, 5.4, 4.1, 1.3, 2.1, 2.4, 5.5, 4.1, 5.5, 5.4, 4.1, 5.4
numTgt=190
UBERList = [[]]
UBERList.append([3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3
UBERList.append([])
UBERList.append(givenAvailability)

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]
m=2880 #Intervals

w1=0.5
w2=0.5

staT=time.time()
plat=platform.system()
    #%%
solution=[]
from pulp import *
given = UBERList[3]#[[1,1,1,1,1],[0,1,1,0,0],[0,0,1,1,0]]
lpCols = len(given[0])
lpRows = len(given)
Rows = []
Cols = []
Track=[]
for i in xrange(1,lpCols+1):
    Cols.append(i)
for i in xrange(1,lpRows+1):
    Rows.append(i)
for i in xrange(0,lpRows):
    if PriorityList[i]==1.1:
        temp=limit[0]
    elif PriorityList[i]==1.2:
        temp=limit[1]
    elif PriorityList[i]==1.3:
        temp=limit[2]
    elif PriorityList[i]==1.4:
        temp=limit[3]
    elif PriorityList[i]==1.5:
        temp=limit[4]
    elif PriorityList[i]==2.1:
```

1

```
        temp=limit[5]
    elif PriorityList[i]==2.2:
        temp=limit[6]
    elif PriorityList[i]==2.3:
        temp=limit[7]
    elif PriorityList[i]==2.4:
        temp=limit[8]
    elif PriorityList[i]==2.5:
        temp=limit[9]
    elif PriorityList[i]==3.1:
        temp=limit[10]
    elif PriorityList[i]==3.2:
        temp=limit[11]
    elif PriorityList[i]==3.3:
        temp=limit[12]
    elif PriorityList[i]==3.4:
        temp=limit[13]
    elif PriorityList[i]==3.5:
        temp=limit[14]
    elif PriorityList[i]==4.1:
        temp=limit[15]
    elif PriorityList[i]==4.2:
        temp=limit[16]
    elif PriorityList[i]==4.3:
        temp=limit[17]
    elif PriorityList[i]==4.4:
        temp=limit[18]
    elif PriorityList[i]==4.5:
        temp=limit[19]
    elif PriorityList[i]==5.1:
        temp=limit[20]
    elif PriorityList[i]==5.2:
        temp=limit[21]
    elif PriorityList[i]==5.3:
        temp=limit[22]
    elif PriorityList[i]==5.4:
        temp=limit[23]
    else:
        temp=limit[24]
    Track.append(temp)

# The prob variable is created to contain the problem data
prob = LpProblem("IP Problem",LpMaximize)

# The problem variables are created
choices = LpVariable.dicts("Choice",(Rows,Cols),0,1,LpInteger)
penalty_pos = LpVariable.dicts("Over",(Rows),0,m,LpInteger)
penalty_neg = LpVariable.dicts("Under",(Rows),0,m,LpInteger)
resetTime = LpVariable.dicts("Choice",(Rows),0,1,LpInteger)

weights=[]
for i in xrange(0,lpRows):
    if PriorityList[i]<2:
        weights.append([1,0.6,0.5])
    elif PriorityList[i]<3:
```

```
        weights.append([1,0.7,0.4])
    elif PriorityList[i]<4:
        weights.append([1,0.8,0.3])
    elif PriorityList[i]<5:
        weights.append([1,0.9,0.2])
    else:
        weights.append([1,0.95,0.1])

AgeWeights=[]
maxAge=max(UBERList[1])*1.0#python does integer devision unless specifying decimals
for i in xrange(0,lpRows):
    if maxAge==0:
        AgeWeights.append(1)
    else:
        AgeWeights.append(15.2279242*UBERList[1][i]/maxAge)

# The objective function is added
prob += lpSum((w1*choices[r][c]*weights[r-1][0] for c in Cols)-w1*penalty_pos[r]*weight

for c in Cols:
    prob += lpSum(choices[r][c] for r in Rows) <= 3,""

for r in Rows:
    prob += lpSum((choices[r][c] for c in Cols)-penalty_pos[r]+penalty_neg[r]) == Track

for r in Rows:
    prob += lpSum(resetTime[r]) <= (choices[r][c] for c in Cols),""

# The starting numbers are entered as constraints
for x in xrange(0,lpRows):
    for y in xrange(0,lpCols):
        if given[x][y] == 0:
            prob += choices[x+1][y+1] == 0,""

# The problem data is written to an .lp file
prob.writeLP("model_lp_out.lp")

# A file called model_lp_out.txt is created/overwritten for writing to
model_lp_out = open('model_lp_out.txt','w')
solCount = 0
#while True:
prob.solve()

# The status of the solution is printed to the screen
print "\n","Status:", LpStatus[prob.status]
# The solution is printed if it was deemed "optimal" i.e met the constraints
if LpStatus[prob.status] == "Optimal":
    solCount += 1
    # The solution is written to the model_lp_out.txt file
    model_lp_out.write("[")
    for r in Rows:
        model_lp_out.write("[")
        pseudoSol=[]
        for c in Cols:
                pseudoSol.append(int(value(choices[r][c])))
```

```
                vee = str(int(value(choices[r][c])))
                model_lp_out.write(vee)
                if (c != lpCols):
                    model_lp_out.write(", ")
                if c == lpCols:
                    model_lp_out.write("]")
                if (c == lpCols) and (r != lpRows):
                    model_lp_out.write(", ")
        solution.append(pseudoSol)
    model_lp_out.write("]")
    model_lp_out.write("\n+-------+-------+-------+\n\n")
    model_lp_out.write("penalty_neg = [")
    for r in Rows:
        vee = str(int(value(penalty_neg[r])))
        model_lp_out.write(vee)
        if r!=lpRows:
            model_lp_out.write(", ")
        pseudoSol=[]
    model_lp_out.write("]\n\n")
    model_lp_out.write("penalty_pos = [")
    for r in Rows:
        vee = str(int(value(penalty_pos[r])))
        model_lp_out.write(vee)
        if r!=lpRows:
            model_lp_out.write(", ")
        pseudoSol=[]
    model_lp_out.write("]\n\n")
    model_lp_out.write("resetTime = [")
    for r in Rows:
        vee = str(int(value(resetTime[r])))
        model_lp_out.write(vee)
        if r!=lpRows:
            model_lp_out.write(", ")
        pseudoSol=[]
    model_lp_out.write("]\n\n")
    print "Objective Value = ", value(prob.objective)
    print "m Value = ", len(UBERList[3])
model_lp_out.close()
# The location of the solutions is give to the user
print "Solutions Written to model_lp_out.txt\n"
#print solution
stopT=time.time()
runTime=(stopT-staT)
print 'Runtime: ' +str(runTime)
```

1.2 Three Sensor Python Code

1.2.1 Data Generation

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script connects with STK and generates access reports for three ground
sensor with all RSO's populated.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import signal
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from itertools import izip as izip, count
import re
import platform
import commands as c
import random

from Targets import tgtList

TS=86400 #total number of seconds in a 24 hour period
repTS=30 #Length of the report timesteps in seconds

print "Started"
staT=time.time()
plat=platform.system()
commands=[]
repCommands=[]
global numInst
dtStart = '21 Jun 2019 00:00:00' #Start date & time
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time

if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
```

1

```
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

numGnd1=1
numGnd2=1
numGnd3=1
numInst=1


 ###########################################################
numTgt=200
numSensors=3 #sum([numGnd1,numGnd2,numGnd3])
if numSensors!=0:

    #%%
    repID='Rep'+str(numInst)
    rstID='RST'+str(numInst)
    #%% Windows directories
    cwd=os.getcwd()
    workPath=cwd
    rstPath = os.path.join(workPath,'RSTfiles_Jun') #Here must be changed to the proper
    curRstPath = os.path.join(rstPath,rstID)
    if os.path.exists(curRstPath):
       rstGlob = os.path.join(curRstPath,'*.rst')
       files=glob.glob(rstGlob)
        #for f in files:
            #os.remove(f)
    else:
        try:
            os.mkdir(rstPath)
        except:
            pass
        os.mkdir(curRstPath)
   rstPath=curRstPath
   repPath = os.path.join(workPath,'Reports_Jun') #Here must be changed to the proper c
   curRepPath = os.path.join(repPath,repID)
    if os.path.exists(curRepPath):
        repGlob = os.path.join(curRepPath,'*.txt')
        files=glob.glob(repGlob)
        #for f in files:
            #os.remove(f)
    else:
        try:
            os.mkdir(repPath)
        except:
            pass
        os.mkdir(curRepPath)
```

```
repPath=curRepPath
os.chdir(rstPath)

#%% Write moon phase report template
f=open('MoonPhase.rst','w')
f.write('stk.v.11.0\nWrittenBy    STK_v11.2.0\n \nBEGIN ReportStyle\n\n')
f.write('BEGIN ClassId\n    Class    Scenario\nEND ClassId\n\n')
f.write('BEGIN Header\n')
f.write('    StyleType    0\n    Date    Yes\n    Name    Yes\n    IsHidden    No\r
f.write('    DescShort    No\n    DescLong    No\n    YLog10    No\n    Y2Log10
f.write('    YUseWholeNumbers    No\n    Y2UseWholeNumbers    No\n    VerticalGridL
f.write('    HorizontalGridLines    No\n    AnnotationType    Spaced\n    NumAnnota
f.write('    NumAngularAnnotations    5\n    ShowYAnnotations    Yes\n    Annotatic
f.write('    BackgroundColor    #ffffff\n    ForegroundColor    #000000\n    Viewab
f.write('    RealTimeMode    No\n    DayLinesStatus    1\n    LegendStatus    1\n
f.write('BEGIN PostProcessor\n    Destination    0\n    Use    0\n    Destination
f.write('    Destination    2\n    Use    0\n    Destination    3\n    Use    0\nEN
f.write('    NumSections    1\nEND Header\n\n')
f.write('BEGIN Section\n    Name    Section 1\n    ClassName    Scenario\n    NameI
f.write('    ExpandMethod    0\n    PropMask    2\n    ShowIntervals    No\n    Num
f.write('BEGIN Line\n    Name    Line 1\n    NumElements    2\n\n')
f.write('BEGIN Element\n    Name    Time\n    IsIndepVar    Yes\n')
f.write('    IndepVarName    Time\n    Title    Time\n    NameInTitle    No\n    Se
f.write('    Type    Moon LunarPhase\n    Element    Time\n    SumAllowedMask    0\
f.write('    DataType    0\n    UnitType    2\n    LineStyle    0\n    LineWidth
f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #000000\n    Fi
f.write('    UseScenUnits    Yes\nEND Element\n\n')
f.write('BEGIN Element\n    Name    Angles-Moon LunarPhase-Angle\n    IsIndepVar
f.write('    IndepVarName    Time\n    Title    Angle\n    NameInTitle    Yes\n
f.write('    Type    Moon LunarPhase\n    Element    Angle\n    SumAllowedMask    1
f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n    LineWidth
f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #000000\n    Fi
f.write('    UseScenUnits    Yes\n')
f.write('END Element\nEND Line\nEND Section\n\n')
f.write('BEGIN LineAnnotations\nEND LineAnnotations\nEND ReportStyle')
f.close()
#%% Write angle report templates for GBT
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            f=open('Tgt_'+str(i)+'_from_Gnd'+str(m)+'_Angles.rst','w')
            f.write('stk.v.11.0\nWrittenBy    STK_v11.2.0\n \nBEGIN ReportStyle\n\r
            f.write('BEGIN ClassId\n    Class    Satellite\nEND ClassId\n\n')
            f.write('BEGIN Header\n')
            f.write('    StyleType    0\n    Date    Yes\n    Name    Yes\n    IsHi
            f.write('    DescShort    No\n    DescLong    No\n    YLog10    No\n
            f.write('    YUseWholeNumbers    No\n    Y2UseWholeNumbers    No\n    \
            f.write('    HorizontalGridLines    No\n    AnnotationType    Spaced\n
            f.write('    NumAngularAnnotations    5\n    ShowYAnnotations    Yes\n
            f.write('    BackgroundColor    #ffffff\n    ForegroundColor    #000000
            f.write('    RealTimeMode    No\n    DayLinesStatus    1\n    LegendSta
            f.write('BEGIN PostProcessor\n    Destination    0\n    Use    0\n    D
            f.write('    Destination    2\n    Use    0\n    Destination    3\n
            f.write('    NumSections    1\nEND Header\n\n')
            f.write('BEGIN Section\n    Name    Section 1\n    ClassName    Satelli
```

```
            f.write('    ExpandMethod    0\n    PropMask    2\n    ShowIntervals
            f.write('BEGIN Line\n    Name    Line 1\n    NumElements    3\n\n')
            f.write('BEGIN Element\n    Name    Time\n    IsIndepVar    Yes\n')
            f.write('    IndepVarName    Time\n    Title    Time\n    NameInTitle
            f.write('    Type    PhaseAngle_Gnd'+str(m)+'\n    Element    Time\n
            f.write('    DataType    0\n    UnitType    2\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\nEND Element\n\n')
            f.write('BEGIN Element\n    Name    Angles-PhaseAngle_Gnd'+str(m)+'-Ang
            f.write('    IndepVarName    Time\n    Title    Solar Phase Angle\n
            f.write('    Type    PhaseAngle_Gnd'+str(m)+'\n    Element    Angle\n
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\nEND Element\n')
            f.write('BEGIN Element\n    Name    Angles-LunarPhaseAngle_Gnd'+str(m)+
            f.write('    IndepVarName    Time\n    Title    Lunar Phase (Obs) Angle
            f.write('    Type    LunarPhaseAngle_Gnd'+str(m)+'\n    Element    Angl
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\n')
            f.write('END Element\nEND Line\nEND Section\n\n')
            f.write('BEGIN LineAnnotations\nEND LineAnnotations\nEND ReportStyle')
            f.close()
#%% Write zenith angle report templates for GBT
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            f=open('Gnd'+str(m)+'_to_Tgt_'+str(i)+'_ZenithAngles.rst','w')
            f.write('stk.v.11.0\nWrittenBy    STK_v11.2.0\n \nBEGIN ReportStyle\n\r
            f.write('BEGIN ClassId\n    Class    Facility\nEND ClassId\n\n')
            f.write('BEGIN Header\n')
            f.write('    StyleType    0\n    Date    Yes\n    Name    Yes\n    IsHi
            f.write('    DescShort    No\n    DescLong    No\n    YLog10    No\n
            f.write('    YUseWholeNumbers    No\n    Y2UseWholeNumbers    No\n    V
            f.write('    HorizontalGridLines    No\n    AnnotationType    Spaced\n
            f.write('    NumAngularAnnotations    5\n    ShowYAnnotations    Yes\n
            f.write('    BackgroundColor    #ffffff\n    ForegroundColor    #000000
            f.write('    RealTimeMode    No\n    DayLinesStatus    1\n    LegendSta
            f.write('BEGIN PostProcessor\n    Destination    0\n    Use    0\n    D
            f.write('    Destination    2\n    Use    0\n    Destination    3\n
            f.write('    NumSections    1\nEND Header\n\n')
            f.write('BEGIN Section\n    Name    Section 1\n    ClassName    Facilit
            f.write('    ExpandMethod    0\n    PropMask    2\n    ShowIntervals
            f.write('BEGIN Line\n    Name    Line 1\n    NumElements    3\n\n')
            f.write('BEGIN Element\n    Name    Time\n    IsIndepVar    Yes\n')
            f.write('    IndepVarName    Time\n    Title    Time\n    NameInTitle
            f.write('    Type    LunarZenith\n    Element    Time\n    SumAllowedMa
            f.write('    DataType    0\n    UnitType    2\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\nEND Element\n\n')
            f.write('BEGIN Element\n    Name    Angles-LunarZenith-Angle\n    IsInc
            f.write('    IndepVarName    Time\n    Title    Lunar Zenith Angle\n
            f.write('    Type    LunarZenith\n    Element    Angle\n    SumAllowedM
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
```

```
            f.write('    UseScenUnits    Yes\nEND Element\n')
            f.write('BEGIN Element\n    Name    Angles-TargetZenith_Gnd'+str(m)+'_t
            f.write('    IndepVarName    Time\n    Title    Target Zenith Angle\n
            f.write('    Type    TargetZenith_Gnd'+str(m)+'_to_Tgt_'+str(i)+'\n
            f.write('    DataType    0\n    UnitType    3\n    LineStyle    0\n
            f.write('    PointSize    0\n    FillPattern    0\n    LineColor    #0
            f.write('    UseScenUnits    Yes\n')
            f.write('END Element\nEND Line\nEND Section\n\n')
            f.write('BEGIN LineAnnotations\nEND LineAnnotations\nEND ReportStyle')
            f.close()


#%% Change directory back to working directory
os.chdir(workPath)
#%% Create Scenario
scenName='Thesis'
concontrol='ConControl / VerboseOn'
commands.append(concontrol) #Please note this script utilizes commands.append not s
unload='Unload / *'
commands.append(unload)
newScen= 'New / Scenario '+str(scenName)
commands.append(newScen)
setTimePeriodStr = 'SetTimePeriod * "' + str(dtStart)+ '" "' + str(dtStop) +'""'
commands.append(setTimePeriodStr)
#%% Create Moon Phase Angle
angStr='VectorTool * CentralBody/Moon Create Angle LunarPhase "Between Vectors" "Ce
commands.append(angStr)
#%% Create Equally Spaced Target Sats
for n in range(1,numTgt+1):
    tgt = str(n)
    newTgt = 'New / */Satellite Tgt_' + str(tgt)
    commands.append(newTgt)
    epoch=tgtList[n-1][0]
    semiMajAxis=tgtList[n-1][1]
    ecc=tgtList[n-1][2]
    inc=tgtList[n-1][3]
    argOfPerigee=tgtList[n-1][4]
    RAAN=tgtList[n-1][5]
    meanAnom=tgtList[n-1][6]
    setStateTgt = ('SetState */Satellite/Tgt_' + str(tgt) +' Classical J2Perturbati
    commands.append(setStateTgt)
    tgtDirLighting="SetConstraint */Satellite/Tgt_"+str(tgt)+" Lighting DirectSun"
    commands.append(tgtDirLighting)

#%% Create Ground-Based Telescopes
print 'Targets created'
loc=[[33.8200 , -106.6600, 1403],[20.7083 , -156.2571, 3052],[-7.3195,72.4229,0]]
for m in range(1,4): #this is iterating through the above list to set the facilitie
    if (eval('numGnd'+str(m)))>0:
        telStr='New / */Facility Gnd'+str(m)
        commands.append(telStr)
        locStr="SetPosition */Facility/Gnd"+str(m)+" Geodetic " + str(loc[m-1][0])
        commands.append(locStr)
        solarExc= "SetConstraint */Facility/Gnd"+str(m)+" LOSSunExclusion 40"
        commands.append(solarExc)
```

```
        lunarExc="SetConstraint */Facility/Gnd"+str(m)+" LOSLunarExclusion 10"
        commands.append(lunarExc)
        lighting="SetConstraint */Facility/Gnd"+str(m)+" Lighting Umbra"
        commands.append(lighting)
        elevationAngle="SetConstraint */Facility/Gnd"+str(m)+" ElevationAngle Min 1
        commands.append(elevationAngle)

#%% Create "To Sensor" vector and Phase Angle for each target/sensor pair.  Also cr
print 'GBTs created'
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            vecStr1='VectorTool * Satellite/Tgt_'+str(i)+' Create Vector ViewVector
            commands.append(vecStr1)
            vecStr2='VectorTool * Facility/Gnd'+str(m)+' Create Vector F2T_Gnd'+str
            commands.append(vecStr2)
            angStr1='VectorTool * Satellite/Tgt_'+str(i)+' Create Angle PhaseAngle_
            commands.append(angStr1)
            angStr2='VectorTool * Facility/Gnd'+str(m)+' Create Angle TargetZenith_
            commands.append(angStr2)

#%% Create Lunar zenith angle for each GBT and phase angle for each target/GBT pair
print 'to sensor vector and phase angle created'
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            angStr1='VectorTool * Satellite/Tgt_'+str(i)+' Create Angle LunarPhaseA
            commands.append(angStr1)
        angStr2='VectorTool * Facility/Gnd'+str(m)+' Create Angle LunarZenith "Betw
        commands.append(angStr2)
#%% Load report styles
print 'Lunar zenith angle created'
os.chdir(rstPath)
loadStrPath = os.path.join(rstPath,'MoonPhase.rst')
moonPath = loadStrPath
loadStr='ReportStyle * Load "'+  str(loadStrPath)  +'"'
commands.append(loadStr)
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            loadStr1Path=os.path.join(rstPath, 'Tgt_'+str(i)+'_from_Gnd'+str(m)+'_A
            loadStr1='ReportStyle * Load "'+  str(loadStr1Path) +'"'
            commands.append(loadStr1)
            loadStr2Path=os.path.join(rstPath, 'Gnd'+str(m)+'_to_Tgt_'+str(i)+'_Zer
            loadStr2='ReportStyle * Load "'+  str(loadStr2Path) +'"'
            commands.append(loadStr2)

#%% Compute access and create access reports for each target/sensor pair
config="ExportConfig / Connection Headers None KeepReportLines Off ShowStartStop Of
repCommands.append(config)
repStrPath =os.path.join(repPath,'MoonPhase.txt')
repStr='ReportCreate * Type Export Style "'+str(moonPath)+'" File "'+  str(repStrPa
repCommands.append(repStr)
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
```

```
        for i in range(1,numTgt+1):
            repStrPath=os.path.join(repPath, 'Tgt_'+str(i)+'_from_Gnd'+str(m)+'_Acc
            repStr='ReportCreate */Satellite/Tgt_'+str(i)+' Type Export Style "Acce
            repCommands.append(repStr)

#%% Send commands to STK
print 'access reports created'
if plat=='Windows':  #Keep an eye for different sections that use the plat=='windo
    repCommands.append('Animate * Reset *')
    for x in commands:
        try:
            s.send(str(x)+'\n')
        except socket.error, e:
            if e.args[0]==10035:
                flag=0
                while flag==0:
                    time.sleep(1)
                    try:
                        s.send(str(x)+'\n')
                        flag=1
                    except:
                        pass
            else:
                print e
                break
   time.sleep(1)
    for y in repCommands:
        try:
            s.send(str(y)+'\n')
        except socket.error, e:
            if e.args[0]==10035:
                flag=0
                while flag==0:
                    time.sleep(1)
                    try:
                        s.send(str(y)+'\n')
                        flag=1
                    except:
                        pass
            else:
                print e
                break
numReps=numSensors*numTgt
#%% Wait for Access reports to show up
os.chdir(repPath)
repCount=0
while repCount<numReps:
    for m in range(1,4):
        if (eval('numGnd'+str(m)))>0:
            fileName='Tgt_'+str(numTgt)+'_from_Gnd'+str(m)+'_AccessRep.txt'
            if os.path.exists(fileName):
                repCount+=1
            else:
                time.sleep(.1)
```

```
   #%% Check to make sure an access occured before creating angle reports
print "All Access Reps found for Instance "+str(numInst)
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            accessName = 'Tgt_'+str(i)+'_from_Gnd'+str(m)+'_AccessRep.txt'
            accessArray = [[str(x) for x in line.strip().split(',')] for line in op
            lenAccess=len(accessArray)
            if lenAccess!=0:
                repStrPath=os.path.join(repPath,'Gnd'+str(m)+'_to_Tgt_'+str(i)+'_Ze
                repStr='ReportCreate */Facility/Gnd'+str(m)+' Type Export Style "Gr
                repStr1Path=os.path.join(repPath,'Tgt_'+str(i)+'_from_Gnd'+str(m)+'
                repStr1='ReportCreate */Satellite/Tgt_'+str(i)+' Type Export Style
                commands.append(repStr)
                commands.append(repStr1)

#%% Create AER reports for each target/sensor pair
for m in range(1,4):
    if (eval('numGnd'+str(m)))>0:
        for i in range(1,numTgt+1):
            repStrPath=os.path.join(repPath,'Tgt_'+str(i)+'_from_Gnd'+str(m)+'_AERR
            repStr='ReportCreate */Facility/Gnd'+str(m)+' Type Export Style AER Fil
            commands.append(repStr)

#%% Send commands to STK
print 'AER reports created for each target/sensor pair'
if plat=='Windows':
    commands.append('Animate * Reset *')
    for x in commands:
        try:
            s.send(str(x)+'\n')
        except socket.error, e:
            if e.args[0]==10035:
                flag=0
                while flag==0:
                    time.sleep(1)
                    try:
                        s.send(str(x)+'\n')
                        flag=1
                    except:
                        pass
            else:
                print e
                break
#%% Wait for AER reports to show up
os.chdir(repPath)
repCount=0
while repCount<numReps:
    for m in range(1,4):
        if (eval('numGnd'+str(m)))>0:
            fileName='Tgt_'+str(numTgt)+'_from_Gnd'+str(m)+'_AERRep.txt'
            if os.path.exists(fileName):
                repCount+=1
            else:
                time.sleep(.1)
```

```
    print "STK finished for Instance "+str(numInst)
    stopT=time.time()
    runTime=(stopT-staT)
    print 'Runtime: ' +str(runTime)
    time.sleep(1)
#%% Clean Up Access reports with bad phase angles
    ObservationDuration=repTS
    print "Instance "+str(numInst)+" done."
```

1.2.2 Base Greedy Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=50

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,3,1,3,1]

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
```

```
    Gnd3D=float(sys.argv[6])


repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

print "Instance "+str(numInst)
print repLocs
print numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
    for g in xrange(0,Intervals):#Creates the list of time steps
        interval=0+g
        AllObservationIntervals.append(interval)
```

```
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)

PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
```

```
            tempCat=4.5
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.1
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.2
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.3
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.4
        else:
            tempCat=5.5
        PriorityList[i]=tempCat

    for j in xrange(1,4):
        BIGList=[]#This list will contain the sensor-target combination lists that indi
        for y in xrange(1,numTgt+1):
            flag=0
            listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
            BIGList.append(listofzeros)
            if j in repLocs:
                tempPath = os.path.join(repPath,'Rep'+str(1))
                os.chdir(tempPath)
                try:
                    fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                    textArray = [[str(x) for x in line.strip().split(',')] for line in
                    if len(textArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    textArray=np.array(textArray)#turns textArray into Numpy Array for
                    T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                    T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                    T=np.array(T)
                    rows = len(T)
                    AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                    AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                    AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                    S=textArray[:,2] #next six lines including this one are the same as
                    S=[i.split('.',1)[0] for i in S]
                    S=np.array(S)

                    AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                    AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                    AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                    IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                    IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                    AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                    AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                    ObservationIntervalsTgt=set()
                    for i in range(0,rows):
                        for l in range(0,AccessStartCount[i,1]):
                            ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                    for i in UBERList[0]:
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
```

```
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list

    PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4

#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    singleObservation=UBERList[w][t][i]
                    minilist.append(singleObservation)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    countervalues=[]
                    for x in indices:#This creates a list of the Counter values for the
                        value=UBERList[1][x]
                        countervalues.append(value)
                    maxofmini=max(countervalues)#finds the max Counter value of the vis
                    max_index=countervalues.index(maxofmini)#finds index of max Counter
                    use_this_index=indices[max_index]#finds index of the Counter that c
                    UBERList[1][use_this_index]=0#resets the counter for the selected t
                    usedindices.append(use_this_index)
                    satCount[use_this_index]=satCount[use_this_index]+1
            for g in xrange(0,numTgt):
                if g in (usedindices):
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[1000,(86400/60.0),0]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
```

```
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
    print '4A: ', count4A
    print '4B: ', count4B
    print '4C: ', count4C
    print '4D: ', count4D
    print '4E: ', count4E
    print '5A: ', count5A
    print '5B: ', count5B
    print '5C: ', count5C
    print '5D: ', count5D
    print '5E: ', count5E
```

```
print '1s: ', count1A+count1B+count1C+count1D+count1E
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3.0) and (PriorityList[i]>2.0):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
```

```
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
```

```
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
ones=[]
for i in xrange(0,len(madeThreshold)):
    if PriorityList[i]<2:
        ones.append(madeThreshold[i])
if ones==[]:
    print 'Total Met Target Threshold of Cat 1: N/A'
```

```
    else:
        print 'Total Met Target Threshold of Cat 1: ', sum(ones)
    twos=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<3) and (PriorityList[i]>2):
            twos.append(madeThreshold[i])
    if twos==[]:
        print 'Total Met Target Threshold of Cat 2: N/A'
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)

else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
    if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
        fitness[0]=100000
        fitness[1]=100000
        fitness[2]=100000
```

1.2.3 SSN Scheduler Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=50

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
```

```
    Gnd3D=float(sys.argv[6])

repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
    for g in xrange(0,Intervals):#Creates the list of time steps
        interval=0+g
        AllObservationIntervals.append(interval)
    UBERList.append(AllObservationIntervals)
```

```
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
```

```
            tempCat=5.1
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.2
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.3
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.4
        else:
            tempCat=5.5
        PriorityList[i]=tempCat

    for j in xrange(1,4):
        BIGList=[]#This list will contain the sensor-target combination lists that indi
        for y in xrange(1,numTgt+1):
            flag=0
            listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
            BIGList.append(listofzeros)
            if j in repLocs:
                tempPath = os.path.join(repPath,'Rep'+str(1))
                os.chdir(tempPath)
                try:
                    fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                    textArray = [[str(x) for x in line.strip().split(',')] for line in
                    if len(textArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    textArray=np.array(textArray)#turns textArray into Numpy Array for
                    T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                    T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                    T=np.array(T)
                    rows = len(T)
                    AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                    AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                    AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                    S=textArray[:,2] #next six lines including this one are the same as
                    S=[i.split('.',1)[0] for i in S]
                    S=np.array(S)

                    AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                    AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                    AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                    IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                    IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                    AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                    AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                    ObservationIntervalsTgt=set()
                    for i in range(0,rows):
                        for l in range(0,AccessStartCount[i,1]):
                            ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                    for i in UBERList[0]:
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list
```

```
    #for testing code
    PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4
#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)

                    zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
                    zenithangleArray=[[str(x) for x in line.strip().split(',')] for lin
                    zenithangleArray=np.array(zenithangleArray)
                    LzenithangleList=zenithangleArray[:,1]
                    TzenithangleList=zenithangleArray[:,2]
                    LzenithAngles=[]
                    TzenithAngles=[]
```

```
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
        #Sky Background, applicable to ground based telescopes only
        I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
        fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
        Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
        Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
        Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
        Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
```

```
                            BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 k
                            Btot=BZ+Bmoon#Sky luminance in lamberts
                            Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                            SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                            #print index[y], len(index)
                            #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                            def detect(rRSO):
                                return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                                #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            res=minimize_scalar(detect, method='golden', options={'xtol
                            rRSO=math.fabs(res.x)
                            Dia=2*rRSO #this is the diameter for a single access for a
                            Size.append(Dia)
                            #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                            SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSC
                        except:
                            pass
#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    #print 'UBERList: ', UBERList[3]
    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                snrlist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    if SNRindex[t][i]>=4.0:
                        singleObservation=UBERList[w][t][i]
                        minilist.append(singleObservation)
                    else:
                        minilist.append(0)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    #print indices
                    indices2=[] #this is the matching list of priorities to those in in
                    indicesSatCount=[] #this is a matching list with the count of how m
                    countervalues=[]
                    shortindex=[]
                    TwoEindex=[]
                    TwoEcountervalues=[]
                    one=[]
                    oneCat=[]
                    oneCount=[]
                    oneQualify=[]
                    two=[]
                    twoCat=[]
                    twoCount=[]
                    twoQualify=[]
                    ThreetoFive=[]
```

```
ThreetoFiveCat=[]
ThreetoFiveCount=[]
snowyFLAG=0
ThreetoFiveQualify=[]
for j in xrange(0,len(minilist)):
    if minilist[j]==1:
        indices2.append(PriorityList[j])
        indicesSatCount.append(satCount[j])
minofmini2=min(indices2)
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<2:
        one.append(indices[j])
        oneCat.append(indices2[j])
        oneCount.append(indicesSatCount[j])
for j in xrange(0,len(oneCount)): #this section is to make a list c
    if oneCat[j]==1.1:
        if oneCount[j]<limit[0]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.2:
        if oneCount[j]<limit[1]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.3:
        if oneCount[j]<limit[2]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.4:
        if oneCount[j]<limit[3]:
            oneQualify.append(one[j])
    else:
        if oneCount[j]<limit[4]:
            oneQualify.append(one[j])
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<3:
        two.append(indices[j])
        twoCat.append(indices2[j])
        twoCount.append(indicesSatCount[j])
for j in xrange(0,len(twoCount)): #this section is to make a list c
    if twoCat[j]==2.1:
        if twoCount[j]<limit[5]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.2:
        if twoCount[j]<limit[6]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.3:
        if twoCount[j]<limit[7]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.4:
        if twoCount[j]<limit[8]:
            twoQualify.append(two[j])
    else:
        if twoCount[j]<limit[9]:
            twoQualify.append(two[j])
for j in xrange(0,len(indices)): #this section is to make a list of
    if indices2[j]>=3:
        ThreetoFive.append(indices[j])
        ThreetoFiveCat.append(indices2[j])
```

```
        ThreetoFiveCount.append(indicesSatCount[j])
for j in xrange(0,len(ThreetoFiveCount)): #this section is to make
    if ThreetoFiveCat[j]==3.1:
        if ThreetoFiveCount[j]<limit[10]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.2:
        if ThreetoFiveCount[j]<limit[11]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.3:
        if ThreetoFiveCount[j]<limit[12]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.4:
        if ThreetoFiveCount[j]<limit[13]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==3.5:
        if ThreetoFiveCount[j]<limit[14]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.1:
        if ThreetoFiveCount[j]<limit[15]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.2:
        if ThreetoFiveCount[j]<limit[16]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.3:
        if ThreetoFiveCount[j]<limit[17]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.4:
        if ThreetoFiveCount[j]<limit[18]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==4.5:
        if ThreetoFiveCount[j]<limit[19]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.1:
        if ThreetoFiveCount[j]<limit[20]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.2:
        if ThreetoFiveCount[j]<limit[21]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.3:
        if ThreetoFiveCount[j]<limit[22]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    elif ThreetoFiveCat[j]==5.4:
        if ThreetoFiveCount[j]<limit[23]:
            ThreetoFiveQualify.append(ThreetoFive[j])
    else:
        if ThreetoFiveCount[j]<limit[24]:
            ThreetoFiveQualify.append(ThreetoFive[j])
if ((minofmini2<2) and (oneQualify!=[])):
    for x in oneQualify:
        value=UBERList[1][x]
        countervalues.append(value)
    maxofmini=max(countervalues)#finds the max Counter value of the
    max_index=countervalues.index(maxofmini)#finds index of max Cou
    use_this_index=oneQualify[max_index]#finds index of the Counter
elif ((minofmini2<2)):
```

```
                        for j in xrange(0,len(indices)):
                            if indices2[j]==minofmini2:
                                shortindex.append(indices[j])
                        for x in shortindex:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=shortindex[max_index]#finds index of the Counter
                    elif ((minofmini2<3) and (twoQualify!=[])):
                        for x in twoQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=twoQualify[max_index]#finds index of the Counter
                    elif ((minofmini2<3)):
                        for j in xrange(0,len(indices)):
                            if indices2[j]==minofmini2:
                                shortindex.append(indices[j])
                        for x in shortindex:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=shortindex[max_index]#finds index of the Counter
                    elif ((minofmini2>=3) and (ThreetoFiveQualify!=[])):
                        for x in ThreetoFiveQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=ThreetoFiveQualify[max_index]#finds index of the
                    else: #if not cat 1.1-2.5 and snowy tables are maxed, treat ALL sat
                        for x in indices:#This creates a list of the Counter values for
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=indices[max_index]#finds index of the Counter th
                    #print use_this_index
                    UBERList[1][use_this_index]=0#resets the counter for the selected t
                    usedindices.append(use_this_index)
                    satCount[use_this_index]=satCount[use_this_index]+1
                    #print SNRindex[use_this_index][i]
            for g in xrange(0,numTgt):
                if g in (usedindices):
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
```

```
else:
    fitness=[(86400/60.0)]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
```

```
print '3C: ', count3C
print '3D: ', count3D
print '3E: ', count3E
print '4A: ', count4A
print '4B: ', count4B
print '4C: ', count4C
print '4D: ', count4D
print '4E: ', count4E
print '5A: ', count5A
print '5B: ', count5B
print '5C: ', count5C
print '5D: ', count5D
print '5E: ', count5E
print '1s: ', count1A+count1B+count1C+count1D+count1E
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
```

```
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
```

```
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
```

```
    madeThreshold=[]
    for i in xrange(0,len(satCount)):
        if satCount[i]>=threshold[i]:
            madeThreshold.append(1)
        else:
            madeThreshold.append(0)
    print 'Total Met Target Threshold All: ', sum(madeThreshold)
    ones=[]
    for i in xrange(0,len(madeThreshold)):
        if PriorityList[i]<2:
            ones.append(madeThreshold[i])
    if ones==[]:
        print 'Total Met Target Threshold of Cat 1: N/A'
    else:
        print 'Total Met Target Threshold of Cat 1: ', sum(ones)
    twos=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<3) and (PriorityList[i]>2):
            twos.append(madeThreshold[i])
    if twos==[]:
        print 'Total Met Target Threshold of Cat 2: N/A'
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)
    print satCount

else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
```

```
fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
#below are the penalty parameters and the gradient of the second tier of the penaly
valMaxSize=75
valMaxLat=90
valMaxCost=30
gradSize=valMaxSize*1.1
gradLat=valMaxLat*1.1
gradCost=valMaxCost*1.1
#Here is where the penalty comes in, after the score is computed then it is accesse
if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
    fitness[0]=100000
    fitness[1]=100000
    fitness[2]=100000
```

1.2.4 Relaxed SSN Scheduler Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=50

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,3,1,3,1]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
```

```
    Gnd3D=float(sys.argv[6])

repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
    for g in xrange(0,Intervals):#Creates the list of time steps
        interval=0+g
        AllObservationIntervals.append(interval)
    UBERList.append(AllObservationIntervals)
```

```
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
```

```
            tempCat=5.1
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.2
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.3
        elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
            tempCat=5.4
        else:
            tempCat=5.5
        PriorityList[i]=tempCat

    for j in xrange(1,4):
        BIGList=[]#This list will contain the sensor-target combination lists that indi
        for y in xrange(1,numTgt+1):
            flag=0
            listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
            BIGList.append(listofzeros)
            if j in repLocs:
                tempPath = os.path.join(repPath,'Rep'+str(1))
                os.chdir(tempPath)
                try:
                    fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                    textArray = [[str(x) for x in line.strip().split(',')] for line in
                    if len(textArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    textArray=np.array(textArray)#turns textArray into Numpy Array for
                    T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                    T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                    T=np.array(T)
                    rows = len(T)
                    AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                    AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                    AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                    S=textArray[:,2] #next six lines including this one are the same as
                    S=[i.split('.',1)[0] for i in S]
                    S=np.array(S)

                    AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                    AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                    AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                    IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                    IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                    AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                    AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                    ObservationIntervalsTgt=set()
                    for i in range(0,rows):
                        for l in range(0,AccessStartCount[i,1]):
                            ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                    for i in UBERList[0]:
                        if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                            BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list
```

```
    #for testing code
PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4

#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)

                    zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
                    zenithangleArray=[[str(x) for x in line.strip().split(',')] for lir
                    zenithangleArray=np.array(zenithangleArray)
                    LzenithangleList=zenithangleArray[:,1]
                    TzenithangleList=zenithangleArray[:,2]
```

```
LzenithAngles=[]
TzenithAngles=[]
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
        #Sky Background, applicable to ground based telescopes only
        I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
        fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
        Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
        Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
```

```
                            Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                            Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                            BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 w
                            Btot=BZ+Bmoon#Sky luminance in lamberts
                            Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                            SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                            #print index[y], len(index)
                            #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                            def detect(rRSO):
                                return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                                #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            res=minimize_scalar(detect, method='golden', options={'xtol
                            rRSO=math.fabs(res.x)
                            Dia=2*rRSO #this is the diameter for a single access for a
                            Size.append(Dia)
                            #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                            SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSC


                        except:
                            pass
#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                snrlist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    if SNRindex[t][i]>=4.0:
                        singleObservation=UBERList[w][t][i]
                        minilist.append(singleObservation)
                    else:
                        minilist.append(0)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    indices2=[] #this is the matching list of priorities to those in in
                    indicesSatCount=[] #this is a matching list with the count of how m
                    countervalues=[]
                    shortindex=[]
                    TwoEindex=[]
                    TwoEcountervalues=[]
                    one=[]
                    oneCat=[]
                    oneCount=[]
                    oneQualify=[]
                    two=[]
                    twoCat=[]
                    twoCount=[]
```

```
twoQualify=[]
ThreetoFive=[]
ThreetoFiveCat=[]
ThreetoFiveCount=[]
ThreetoFiveQualify=[]
for j in xrange(0,len(minilist)):
    if minilist[j]==1:
        indices2.append(PriorityList[j])
        indicesSatCount.append(satCount[j])
minofmini2=min(indices2)
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<2:
        one.append(indices[j])
        oneCat.append(indices2[j])
        oneCount.append(indicesSatCount[j])
for j in xrange(0,len(oneCount)): #this section is to make a list c
    if oneCat[j]==1.1:
        if oneCount[j]<limit[0]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.2:
        if oneCount[j]<limit[1]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.3:
        if oneCount[j]<limit[2]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.4:
        if oneCount[j]<limit[3]:
            oneQualify.append(one[j])
    else:
        if oneCount[j]<limit[4]:
            oneQualify.append(one[j])
for j in xrange(0,len(indices)): #this section is to make lists of
    if indices2[j]<3:
        two.append(indices[j])
        twoCat.append(indices2[j])
        twoCount.append(indicesSatCount[j])
for j in xrange(0,len(twoCount)): #this section is to make a list c
    if twoCat[j]==2.1:
        if twoCount[j]<limit[5]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.2:
        if twoCount[j]<limit[6]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.3:
        if twoCount[j]<limit[7]:
            twoQualify.append(two[j])
    elif twoCat[j]==2.4:
        if twoCount[j]<limit[8]:
            twoQualify.append(two[j])
    else:
        if twoCount[j]<limit[9]:
            twoQualify.append(two[j])
for j in xrange(0,len(indices)): #this section is to make a list of
    if indices2[j]>=3:
        ThreetoFive.append(indices[j])
```

```
            ThreetoFiveCat.append(indices2[j])
            ThreetoFiveCount.append(indicesSatCount[j])
    for j in xrange(0,len(ThreetoFiveCount)): #this section is to make
        if ThreetoFiveCat[j]==3.1:
            if ThreetoFiveCount[j]<limit[10]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.2:
            if ThreetoFiveCount[j]<limit[11]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.3:
            if ThreetoFiveCount[j]<limit[12]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.4:
            if ThreetoFiveCount[j]<limit[13]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.5:
            if ThreetoFiveCount[j]<limit[14]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.1:
            if ThreetoFiveCount[j]<limit[15]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.2:
            if ThreetoFiveCount[j]<limit[16]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.3:
            if ThreetoFiveCount[j]<limit[17]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.4:
            if ThreetoFiveCount[j]<limit[18]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.5:
            if ThreetoFiveCount[j]<limit[19]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.1:
            if ThreetoFiveCount[j]<limit[20]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.2:
            if ThreetoFiveCount[j]<limit[21]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.3:
            if ThreetoFiveCount[j]<limit[22]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.4:
            if ThreetoFiveCount[j]<limit[23]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        else:
            if ThreetoFiveCount[j]<limit[24]:
                ThreetoFiveQualify.append(ThreetoFive[j])
    if ((minofmini2<2) and (oneQualify!=[])):
        for x in oneQualify:
            value=UBERList[1][x]
            countervalues.append(value)
        maxofmini=max(countervalues)#finds the max Counter value of the
        max_index=countervalues.index(maxofmini)#finds index of max Cou
        use_this_index=oneQualify[max_index]#finds index of the Counter
```

```
                    elif ((minofmini2<3) and (twoQualify!=[])):
                        for x in twoQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=twoQualify[max_index]#finds index of the Counter
                    elif ((minofmini2>=3) and (ThreetoFiveQualify!=[])):
                        for x in ThreetoFiveQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=ThreetoFiveQualify[max_index]#finds index of the
                    else: #if not cat 1.1-2.5 and snowy tables are maxed, treat ALL sat
                        for x in indices:#This creates a list of the Counter values for
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=indices[max_index]#finds index of the Counter th
                    UBERList[1][use_this_index]=0#resets the counter for the selected t
                    usedindices.append(use_this_index)
                    satCount[use_this_index]=satCount[use_this_index]+1
                for g in xrange(0,numTgt):
                    if g in (usedindices):
                        continue
                    UBERList[w][g][i]=0#changes all but the selected targets selection indi

        #%%
        stopT=time.time()
        runTime=(stopT-staT)
    # For efficiency, if the null architecture shows up, skip the simulation and give it th
    # The "time.sleep" command ensures that the null instance doesn't start the failed node
    else:
        fitness=[(86400/60.0)]

    count1A=PriorityList.count(1.1)
    count1B=PriorityList.count(1.2)
    count1C=PriorityList.count(1.3)
    count1D=PriorityList.count(1.4)
    count1E=PriorityList.count(1.5)
    count2A=PriorityList.count(2.1)
    count2B=PriorityList.count(2.2)
    count2C=PriorityList.count(2.3)
    count2D=PriorityList.count(2.4)
    count2E=PriorityList.count(2.5)
    count3A=PriorityList.count(3.1)
    count3B=PriorityList.count(3.2)
    count3C=PriorityList.count(3.3)
    count3D=PriorityList.count(3.4)
    count3E=PriorityList.count(3.5)
    count4A=PriorityList.count(4.1)
    count4B=PriorityList.count(4.2)
    count4C=PriorityList.count(4.3)
```

```
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
    print '4A: ', count4A
    print '4B: ', count4B
    print '4C: ', count4C
    print '4D: ', count4D
    print '4E: ', count4E
    print '5A: ', count5A
    print '5B: ', count5B
    print '5C: ', count5C
    print '5D: ', count5D
    print '5E: ', count5E
    print '1s: ', count1A+count1B+count1C+count1D+count1E
    print '2s: ', count2A+count2B+count2C+count2D+count2E
    print '3s: ', count3A+count3B+count3C+count3D+count3E
    print '4s: ', count4A+count4B+count4C+count4D+count4E
    print '5s: ', count5A+count5B+count5C+count5D+count5E

    print 'Results: '
    print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
```

```
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
```

```
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
```

```
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
ones=[]
for i in xrange(0,len(madeThreshold)):
    if PriorityList[i]<2:
        ones.append(madeThreshold[i])
if ones==[]:
    print 'Total Met Target Threshold of Cat 1: N/A'
else:
    print 'Total Met Target Threshold of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(madeThreshold)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(madeThreshold[i])
if twos==[]:
    print 'Total Met Target Threshold of Cat 2: N/A'
```

```
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)
else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
    if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
        fitness[0]=100000
        fitness[1]=100000
        fitness[2]=100000
```

1.2.5 Relaxed SSN Scheduler Model with Spacing

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=50

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

#number of intervals between NEEDing to look at it (if I just looked at it not too long
spread=60

arch=[3,1,3,1,3,1]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
```

```
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])


repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
```

```
for g in xrange(0,Intervals):#Creates the list of time steps
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[0 for x in range(Intervals)] for y in range(numTgt)]

#Manual setting of priority list (use this or random)
#PriorityList=[1,2,3,4,5]
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    random5=random.randint(1,5)
    if random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        if random5==1:
            tempCat=1.1
        elif random5==2:
            tempCat=1.2
        elif random5==3:
            tempCat=1.3
        elif random5==4:
            tempCat=1.4
        elif random5==5:
            tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
```

```
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat


for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')] for line in
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
                rows = len(T)
                AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                S=textArray[:,2] #next six lines including this one are the same as
                S=[i.split('.',1)[0] for i in S]
                S=np.array(S)

                AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                ObservationIntervalsTgt=set()
                for i in range(0,rows):
```

```
                    for l in range(0,AccessStartCount[i,1]):
                        ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                for i in UBERList[0]:
                    if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                        BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list

    PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4

#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)
```

```
zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
zenithangleArray=[[str(x) for x in line.strip().split(',')] for lir
zenithangleArray=np.array(zenithangleArray)
LzenithangleList=zenithangleArray[:,1]
TzenithangleList=zenithangleArray[:,2]
LzenithAngles=[]
TzenithAngles=[]
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-b
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
```

```
                            #Sky Background, applicable to ground based telescopes only
                            I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
                            fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
                            Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
                            Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
                            Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                            Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                            BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 k
                            Btot=BZ+Bmoon#Sky luminance in lamberts
                            Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                            SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                            #print index[y], len(index)
                            #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                            def detect(rRSO):
                                return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                                #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            res=minimize_scalar(detect, method='golden', options={'xtol
                            rRSO=math.fabs(res.x)
                            Dia=2*rRSO #this is the diameter for a single access for a
                            Size.append(Dia)
                            #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                            SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRSO
                        except:
                            pass

#%%
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

    for i in xrange(0,Intervals):
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        Target_order=[]
        for w in range(3,6):
            posObs=eval('numGnd'+str(w-2))
            usedindices=[]
            for pos in range(0,posObs):
                minilist=[]
                snrlist=[]
                for t in xrange(0,numTgt):#this section creates a "minilist" that is a
                    if SNRindex[t][i]>=4.0:
                        singleObservation=UBERList[w][t][i]
                        minilist.append(singleObservation)
                    else:
                        minilist.append(0)
                if sum(minilist)!=0:
                    indices=[k for k, x in enumerate(minilist) if x==1] #This line crea
                    indices2=[] #this is the matching list of priorities to those in in
                    indicesSatCount=[] #this is a matching list with the count of how m
                    SPREADindices=[]
                    SPREADindices2=[]
                    SPREADindicesSatCount=[]
                    countervalues=[]
                    SPREADcountervalues=[]
                    shortindex=[]
```

```
TwoEindex=[]
TwoEcountervalues=[]
one=[]
oneCat=[]
oneCount=[]
oneQualify=[]
two=[]
twoCat=[]
twoCount=[]
twoQualify=[]
ThreetoFive=[]
ThreetoFiveCat=[]
ThreetoFiveCount=[]
ThreetoFiveQualify=[]
for j in xrange(0,len(minilist)):
    if minilist[j]==1:
        indices2.append(PriorityList[j])
        indicesSatCount.append(satCount[j])
minofmini2=min(indices2)
for x in indices:#This section is creating a list of those visible
    value=UBERList[1][x]
    SPREADcountervalues.append(value)
for j in xrange(0,len(SPREADcountervalues)):
    if SPREADcountervalues[j]>spread:
        SPREADindices.append(indices[j])
        SPREADindices2.append(indices2[j])
        SPREADindicesSatCount.append(indicesSatCount[j])
for j in xrange(0,len(SPREADindices)): #this section is to make lis
    if SPREADindices2[j]<2:
        one.append(SPREADindices[j])
        oneCat.append(SPREADindices2[j])
        oneCount.append(SPREADindicesSatCount[j])
for j in xrange(0,len(oneCount)): #this section is to make a list c
    if oneCat[j]==1.1:
        if oneCount[j]<limit[0]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.2:
        if oneCount[j]<limit[1]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.3:
        if oneCount[j]<limit[2]:
            oneQualify.append(one[j])
    elif oneCat[j]==1.4:
        if oneCount[j]<limit[3]:
            oneQualify.append(one[j])
    else:
        if oneCount[j]<limit[4]:
            oneQualify.append(one[j])
for j in xrange(0,len(SPREADindices)): #this section is to make lis
    if SPREADindices2[j]<3:
        two.append(SPREADindices[j])
        twoCat.append(SPREADindices2[j])
        twoCount.append(SPREADindicesSatCount[j])
for j in xrange(0,len(twoCount)): #this section is to make a list c
    if twoCat[j]==2.1:
```

```
            if twoCount[j]<limit[5]:
                twoQualify.append(two[j])
        elif twoCat[j]==2.2:
            if twoCount[j]<limit[6]:
                twoQualify.append(two[j])
        elif twoCat[j]==2.3:
            if twoCount[j]<limit[7]:
                twoQualify.append(two[j])
        elif twoCat[j]==2.4:
            if twoCount[j]<limit[8]:
                twoQualify.append(two[j])
        else:
            if twoCount[j]<limit[9]:
                twoQualify.append(two[j])
    for j in xrange(0,len(SPREADindices)): #this section is to make a l
        if SPREADindices2[j]>=3:
            ThreetoFive.append(SPREADindices[j])
            ThreetoFiveCat.append(SPREADindices2[j])
            ThreetoFiveCount.append(SPREADindicesSatCount[j])
    for j in xrange(0,len(ThreetoFiveCount)): #this section is to make
        if ThreetoFiveCat[j]==3.1:
            if ThreetoFiveCount[j]<limit[10]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.2:
            if ThreetoFiveCount[j]<limit[11]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.3:
            if ThreetoFiveCount[j]<limit[12]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.4:
            if ThreetoFiveCount[j]<limit[13]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==3.5:
            if ThreetoFiveCount[j]<limit[14]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.1:
            if ThreetoFiveCount[j]<limit[15]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.2:
            if ThreetoFiveCount[j]<limit[16]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.3:
            if ThreetoFiveCount[j]<limit[17]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.4:
            if ThreetoFiveCount[j]<limit[18]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==4.5:
            if ThreetoFiveCount[j]<limit[19]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.1:
            if ThreetoFiveCount[j]<limit[20]:
                ThreetoFiveQualify.append(ThreetoFive[j])
        elif ThreetoFiveCat[j]==5.2:
            if ThreetoFiveCount[j]<limit[21]:
```

```
                            ThreetoFiveQualify.append(ThreetoFive[j])
                        elif ThreetoFiveCat[j]==5.3:
                            if ThreetoFiveCount[j]<limit[22]:
                                ThreetoFiveQualify.append(ThreetoFive[j])
                        elif ThreetoFiveCat[j]==5.4:
                            if ThreetoFiveCount[j]<limit[23]:
                                ThreetoFiveQualify.append(ThreetoFive[j])
                        else:
                            if ThreetoFiveCount[j]<limit[24]:
                                ThreetoFiveQualify.append(ThreetoFive[j])
                    if ((minofmini2<2) and (oneQualify!=[])):
                        for x in oneQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=oneQualify[max_index]#finds index of the Counter
                    elif ((minofmini2<3) and (twoQualify!=[])):
                        for x in twoQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=twoQualify[max_index]#finds index of the Counter
                    elif ((minofmini2>=3) and (ThreetoFiveQualify!=[])):
                        for x in ThreetoFiveQualify:
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=ThreetoFiveQualify[max_index]#finds index of the
                    else: #if not cat 1.1-2.5 and snowy tables are maxed, treat ALL sat
                        for x in indices:#This creates a list of the Counter values for
                            value=UBERList[1][x]
                            countervalues.append(value)
                        maxofmini=max(countervalues)#finds the max Counter value of the
                        max_index=countervalues.index(maxofmini)#finds index of max Cou
                        use_this_index=indices[max_index]#finds index of the Counter th
                    UBERList[1][use_this_index]=0#resets the counter for the selected t
                    #print UBERList[1]
                    usedindices.append(use_this_index)
                    satCount[use_this_index]=satCount[use_this_index]+1
            for g in xrange(0,numTgt):
                if g in (usedindices):
                    continue
                UBERList[w][g][i]=0#changes all but the selected targets selection indi

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]
```

```
count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
```

```
print '4A: ', count4A
print '4B: ', count4B
print '4C: ', count4C
print '4D: ', count4D
print '4E: ', count4E
print '5A: ', count5A
print '5B: ', count5B
print '5C: ', count5C
print '5D: ', count5D
print '5E: ', count5E
print '1s: ', count1A+count1B+count1C+count1D+count1E
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
```

```
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
```

```
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
```

```
            madeThreshold.append(1)
        else:
            madeThreshold.append(0)
    print 'Total Met Target Threshold All: ', sum(madeThreshold)
    ones=[]
    for i in xrange(0,len(madeThreshold)):
        if PriorityList[i]<2:
            ones.append(madeThreshold[i])
    if ones==[]:
        print 'Total Met Target Threshold of Cat 1: N/A'
    else:
        print 'Total Met Target Threshold of Cat 1: ', sum(ones)
    twos=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<3) and (PriorityList[i]>2):
            twos.append(madeThreshold[i])
    if twos==[]:
        print 'Total Met Target Threshold of Cat 2: N/A'
    else:
        print 'Total Met Target Threshold of Cat 2: ', sum(twos)
    threes=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<4) and (PriorityList[i]>3):
            threes.append(madeThreshold[i])
    if threes==[]:
        print 'Total Met Target Threshold of Cat 3: N/A'
    else:
        print 'Total Met Target Threshold of Cat 3: ', sum(threes)
    fours=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]<5) and (PriorityList[i]>4):
            fours.append(madeThreshold[i])
    if fours==[]:
        print 'Total Met Target Threshold of Cat 4: N/A'
    else:
        print 'Total Met Target Threshold of Cat 4: ', sum(fours)
    fives=[]
    for i in xrange(0,len(madeThreshold)):
        if (PriorityList[i]>5):
            fives.append(madeThreshold[i])
    if fives==[]:
        print 'Total Met Target Threshold of Cat 5: N/A'
    else:
        print 'Total Met Target Threshold of Cat 5: ', sum(fives)
else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
```

```
gradSize=valMaxSize*1.1
gradLat=valMaxLat*1.1
gradCost=valMaxCost*1.1
#Here is where the penalty comes in, after the score is computed then it is accesse
if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
    fitness[0]=100000
    fitness[1]=100000
    fitness[2]=100000
```

1.2.6 Integer Program Setup

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=50

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
```

1

```
probCat5A=0.1
probCat5B=0.1
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

#number of intervals between NEEDing to look at it (if I just looked at it not too long
spread=2

arch=[3,1,3,1,3,1]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
```

```
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])

repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

        #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Cc
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20076
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
    for g in xrange(0,Intervals):#Creates the list of time steps
```

```
    interval=0+g
    AllObservationIntervals.append(interval)
UBERList.append(AllObservationIntervals)
Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
UBERList.append(Counter)
IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
UBERList.append(IDCounter)
PriorityList=[0]*numTgt #creates and initializes a list of priorities
satCount=[0]*numTgt #this creates and initializes the list which holds how many tim
SNRindex = [[[0 for x in range(Intervals)] for y in range(numTgt)]for z in range(nu

#Manual setting of priority list (use this or random)
#This section assigns priority categories to each sat in list. assignment is based
for i in xrange(0,numTgt):
    random100=random.randint(1,100)
    if random100<=probCat1A*100:
        tempCat=1.1
    elif random100<=(probCat1A+probCat1B)*100:
        tempCat=1.2
    elif random100<=(probCat1A+probCat1B+probCat1C)*100:
        tempCat=1.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D)*100:
        tempCat=1.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E)*100:
        tempCat=1.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A)*1
        tempCat=2.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=2.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=3.5
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.4
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=4.5
```

```
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.1
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.2
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.3
    elif random100<=(probCat1A+probCat1B+probCat1C+probCat1D+probCat1E+probCat2A+pr
        tempCat=5.4
    else:
        tempCat=5.5
    PriorityList[i]=tempCat

for j in xrange(1,4):
    BIGList=[]#This list will contain the sensor-target combination lists that indi
    for y in xrange(1,numTgt+1):
        flag=0
        listofzeros=[0]*(Intervals) #creates a list of zeros, Intervals long, for e
        BIGList.append(listofzeros)
        if j in repLocs:
            tempPath = os.path.join(repPath,'Rep'+str(1))
            os.chdir(tempPath)
            try:
                fileName = 'Tgt_' +str(y)+'_from_Gnd'+str(j)+'_AccessRep.txt'
                textArray = [[str(x) for x in line.strip().split(',')] for line in
                if len(textArray)>0:
                    flag=1
            except:
                pass
            if flag==1:
                textArray=np.array(textArray)#turns textArray into Numpy Array for
                T=textArray[:,1] #pulls element 1 (Access Start Time) out of each r
                T=[i.split('.',1)[0] for i in T]#removes the milliseconds from the
                T=np.array(T)
                rows = len(T)
                AccessStartDates=[datetime.datetime.strptime(str(T[i]),'%d %b %Y %H
                AccessStartTime=[timedelta.total_seconds(AccessStartDates[i]) for i
                AccessStartTime=np.array([int(ObservationDuration*math.ceil(i/Obser
                S=textArray[:,2] #next six lines including this one are the same as
                S=[i.split('.',1)[0] for i in S]
                S=np.array(S)

                AccessStopDates=[datetime.datetime.strptime(str(S[i]),'%d %b %Y %H:
                AccessStopTime=[timedelta.total_seconds(AccessStopDates[i]) for i i
                AccessStopTime=np.array([int(ObservationDuration*math.floor(i/Obser
                IntervalDuration=AccessStopTime-AccessStartTime#self explanatory
                IntervalCount=np.array([(i/ObservationDuration) for i in IntervalDu
                AccessStartCount=np.vstack((AccessStartTime,IntervalCount)).reshape
                AccessStartCount=np.reshape(AccessStartCount,(-1,2))
                ObservationIntervalsTgt=set()
                for i in range(0,rows):
                    for l in range(0,AccessStartCount[i,1]):
                        ObservationIntervalsTgt.add(AccessStartCount[i,0]/Observati
                for i in UBERList[0]:
                    if i in ObservationIntervalsTgt and i in clearSkySetList[j-1]:
                        BIGList[y-1][i]=1 #puts 1's in the list of zeros created ea
```

```
        UBERList.append(BIGList)# this list contains "Intervals," "Counter," and a list

        #for testing code
    PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4
    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3

#
    MoonPhasefileName='MoonPhase.txt'
    MoonPhase=open(MoonPhasefileName, 'r').readline().split(',')
    lunarphase=np.float(MoonPhase[1])

    AtmosTran=[0.794674,0.908067,0.900025,0.929173,0.948,0.914859,0.913138,0.85661,0.91
    from itertools import izip as izip, count
    Latency=[]
    Max=[]
    Min=[]
    Size=[]
    AllTargetsIndices=[]
    for j in xrange(0,numTgt):
        #indices=[]
        Diff=[]
        for i in xrange(3,6):
            index=[k for k, x in izip(count(),UBERList[i][j]) if x==1] #USE THIS TO LOC
            flag=0
            if i in range(3,12) and (eval('numGnd'+str(i-2)))>0:
                tempPath = os.path.join(repPath,'Rep'+str(numInst))
                os.chdir(tempPath)
                try:
                    gndrangefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AERRep.tx
                    gndrangeArray = [[str(gnd) for gnd in line.strip().split(',')] for
                    if len(gndrangeArray)>0:
                        flag=1
                except:
                    pass
                if flag==1:
                    gndrangeArray=np.array(gndrangeArray)
                    gndT=gndrangeArray[:,0]
                    gndT=[lin.split('.',1)[0] for lin in gndT]
                    gndT=np.array(gndT)
                    rows = len(gndT)
                    ObsStartTime=[int(timedelta.total_seconds(datetime.datetime.strptim
                    #print ObsStartTime
                    ObsStartIntervals=[int(math.floor(OSI/ObservationDuration)) for OSI
                    R=gndrangeArray[:,3]
                    R=[int(ran.split('.',1)[0]) for ran in R]
                    # This section determines the indices of ObsStartIntervals (which c
                    Ranges=[]
                    for k in index:
                        Rindex=bisect.bisect_left(ObsStartIntervals, k) #finds index of
                        Range=R[Rindex]
                        Ranges.append(Range)

                    zenithanglefileName= 'Gnd'+str(i-2)+'_to_Tgt_'+str(j+1)+'_ZenithAng
                    zenithangleArray=[[str(x) for x in line.strip().split(',')] for lir
                    zenithangleArray=np.array(zenithangleArray)
```

```
LzenithangleList=zenithangleArray[:,1]
TzenithangleList=zenithangleArray[:,2]
LzenithAngles=[]
TzenithAngles=[]
for k in index:
    try:
        LzA=float(LzenithangleList[k])
        TzA=float(TzenithangleList[k])
        LzenithAngles.append(LzA)
        TzenithAngles.append(TzA)
    except:
        pass
anglefileName= 'Tgt_' +str(j+1)+'_from_Gnd'+str(i-2)+'_AngleRep.txt
angleArray=[[str(x) for x in line.strip().split(',')] for line in c
angleArray=np.array(angleArray)
phaseangleList=angleArray[:,1]
lunarphaseangleList=angleArray[:,2]
PhaseAngles=[]
lunarPhaseAngles=[]
for k in index:
    try:
        PhsA=float(phaseangleList[k])
        LPhsA=float(lunarphaseangleList[k])
        PhaseAngles.append(PhsA)
        lunarPhaseAngles.append(LPhsA)
    except:
        pass
#print len(index)
for y in range(0, len(index)):
    #print y
    try:
        Tint=1.0
        Range=Ranges[y]*1000# this will come from STK
        phaseangle=math.radians(PhaseAngles[y]) #(rad) This will cc
        lunarobsang=180-lunarPhaseAngles[y] #(degrees) this value w
        lunarzenith=LzenithAngles[y] #(degrees) this value will com
        targetzenith=TzenithAngles[y]
        #Variable from design scripted in python
        D=eval('Gnd'+str(i-2)+'D') #outside aperture diameter, vari
        d=D*0.3 #obscuration diameter, variable from system design
        AT=AtmosTran[i-3] #zenith path transmission,from LEEDR simu
        kmag=-(2.5*math.log10(AT)) #zenith path extinction in astrc
        #General Calcs
        focalLength=2*D #Assumes a fast, but not quite state of the
        IFOV=9.6963*10**-6 #radians, this is applicable to ground-t
        pixPitch=focalLength*IFOV
        Npix=math.ceil(((7.2722*10**-5)*Tint/IFOV)*2)
        #it is motion of the GEO belt relative to the star backgrou
        #(worstcase when the image is between two pixel rows)
        CoRef=(2.0/3.0)*(refl/(math.pi**2))*(math.sin(phaseangle)+(
        Arcvr=(math.pi*(D/2)**2)-(math.pi*(d/2)**2) #sensor area, c
        pathtrans=AT**(1/(math.cos(math.radians(targetzenith))))
        #Sky Background, applicable to ground based telescopes only
        I=10**(-.4*(3.84+0.026*math.fabs(lunarphase)+4*10**-9*lunar
        fLOA=10**5.36*(1.06+(math.cos(math.radians(lunarobsang)))**
```

```
                            Zdistmoon=(1-0.96*(math.sin(math.radians(lunarzenith)))**2)
                            Zdist=(1-0.96*(math.sin(math.radians(targetzenith)))**2)**(
                            Bmoon=fLOA*I*10**(-.4*kmag*Zdistmoon)*(1-10**(-.4*kmag*Zdis
                            Bzen=(123.73*10**-9)#zenith sky irradiance in Lamberts
                            BZ=Bzen*10**(-.4*kmag*(Zdist-1))*Zdist #lamberts (4.66047 k
                            Btot=BZ+Bmoon#Sky Luminance in Lamberts
                            Radsky=4.66047*Btot#sky radiance Watts/(m^2*sr)
                            SkySignal=(Radsky*Arcvr*opttrans*QE*avgwavelength*Tint*IFOV
                            #print index[y], len(index)
                            #SNRindex[j][index[y]] = math.fabs(((VisSolflux*math.pi*rRS
                            def detect(rRSO):
                                return math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                                #print math.fabs(((VisSolflux*math.pi*rRSO**2*CoRef*pat
                            res=minimize_scalar(detect, method='golden', options={'xtol
                            rRSO=math.fabs(res.x)
                            Dia=2*rRSO #this is the diameter for a single access for a
                            Size.append(Dia)
                            #print j, index[y], math.fabs(((VisSolflux*math.pi*rRSO**2*
                            SNRindex[i-3][j][index[y]] = math.fabs(((VisSolflux*math.pi
                        except:
                            pass

    #print len(UBERList[3][0])
    #print UBERList[3]
    givenAvailability=[]
    for z in xrange(3,6):
        tempSensor=[]
        for x in xrange(0,len(UBERList[z])):
            temp=[]
            for y in xrange(0,len(UBERList[z][0])):
                if (UBERList[z][x][y]==1): #and SNRindex[x][y]>=6.0:
                    temp.append(1)
                else:
                    temp.append(0)
            tempSensor.append(temp)
        givenAvailability.append(tempSensor)

    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]
model_lp_out = open('model_lp_out.txt','w')


# The status of the solution is printed to the screen
print "\n","Status:"
model_lp_out.write("ANYTHING")
model_lp_out.write("]")
model_lp_out.write("\n+-------+-------+-------+\n\n")
print "Objective Value = "
print "m Value = "
model_lp_out.close()
```

```
countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]

plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)

else:
    workPath = os.environ['LOC']
    workSpace = os.environ['WORKDIR']
    repPath = os.path.join(workSpace,'Reports_Jan')
    scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
    os.chdir(scorePath)
    fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
    #below are the penalty parameters and the gradient of the second tier of the penaly
    valMaxSize=75
    valMaxLat=90
    valMaxCost=30
    gradSize=valMaxSize*1.1
    gradLat=valMaxLat*1.1
    gradCost=valMaxCost*1.1
    #Here is where the penalty comes in, after the score is computed then it is accesse
    if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
        fitness[0]=100000
        fitness[1]=100000
        fitness[2]=100000
```

1.2.7 Integer Program Evaluation

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will evaluate the IP schedule completed in previous scripts.
"""

#import socket #imports a python class needed to establish TCP/IP
import sys
import os
import glob
import math
import time
import socket
import datetime
from datetime import timedelta
import numpy as np
import bisect
from scipy.optimize import minimize_scalar
import re
import platform
import commands
from clearSky import clearSkySetList
import random

TS=86400
repTS=30
numTgt=50

#Category Probabilities
#these should sum to 1
probCat1A=0.002
probCat1B=0.002
probCat1C=0.002
probCat1D=0.002
probCat1E=0.002
probCat2A=0.0375
probCat2B=0.0375
probCat2C=0.0375
probCat2D=0.0375
probCat2E=0.04
probCat3A=0.01
probCat3B=0.01
probCat3C=0.01
probCat3D=0.01
probCat3E=0.01
probCat4A=0.05
probCat4B=0.05
probCat4C=0.05
probCat4D=0.05
probCat4E=0.05
probCat5A=0.1
probCat5B=0.1
```

1

```
probCat5C=0.1
probCat5D=0.1
probCat5E=0.1

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]

arch=[3,1,0,0,0,0]#3,1,3,1,3,1] #this is how many sensors or how many objects can be si

staT=time.time()
plat=platform.system()

dtStart = '21 Jun 2019 00:00:00' #Start date & time remember to match these up with the
dtStop= '22 Jun 2019 00:00:00'  #Stop date & time
trial_num = 'Trial_10'  #This indicates what num trial is being run when this script is
if plat=='Windows':
    import winsound
    HOST = socket.gethostname()
    PORT = 5001 # This is the default port identified by AGI
    s = None # s is a socket object that we will use to pass info from our Python progr
    for res in socket.getaddrinfo(HOST, PORT, socket.AF_UNSPEC, socket.SOCK_STREAM):
        af, socktype, proto, canonname, sa = res
        try:
            s = socket.socket(af, socktype, proto)
        except socket.error, msg:
            s = None
            continue
        try:
            s.connect(sa)
        except socket.error, msg:
            s.close()
            s = None
            continue
        break
    if s is None:
        print 'Could not open socket - Please start STK or STKEngine first'
        sys.exit(1)
    s.setblocking(False)

    numGnd1=int(arch[0])
    Gnd1D=float(arch[1])
    numGnd2=int(arch[2])
    Gnd2D=float(arch[3])
    numGnd3=int(arch[4])
    Gnd3D=float(arch[5])
    numInst=1

else:
    numGnd1=int(sys.argv[1])
    Gnd1D=float(sys.argv[2])
    numGnd2=int(sys.argv[3])
    Gnd2D=float(sys.argv[4])
    numGnd3=int(sys.argv[5])
    Gnd3D=float(sys.argv[6])
```

```
repLocs=[]
if numGnd1>0:
    repLocs.append(1)
if numGnd2>0:
    repLocs.append(2)
if numGnd3>0:
    repLocs.append(3)

#print "Instance "+str(numInst)
print "Number of Ground Sensors: ", repLocs
print "Number of targets: ", numTgt
if sum([numGnd1,numGnd2,numGnd3])!=0:

    #%%
    repID='Rep'+str(numInst)
    #%% Windows directories
    workPath=os.getcwd()
    repPath = os.path.join(workPath,'Reports_Jun')
    #%% HPC directories
    if plat!='Windows':
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(repPath)

#         #%%
    ScenarioDuration=timedelta.total_seconds(datetime.datetime.strptime(dtStop,'%d %b %
    ObservationDuration=repTS
    Intervals=int((ScenarioDuration*86400)/ObservationDuration) # number of IntervalDur
    SpeedOLight=2.998*10**8; #(m/s) speed of light
    PlanckConst=6.626*10**(-34) #(J/s) Planck's constant
    magSolsqas=-10.7#apparent magnitude of sun per square arcsecond
    SolRad=3144586# W/(m^2*str), SolLum*(1/628) to convert from cd/m^2 to W/(m^2*str)
    spaceVM=22# /arsec^2. Source: "Ground Optical Signal Processing Architecture for Co
    spaceRadsky=SolRad*10**(0.4*(magSolsqas-spaceVM))#space sky radiance, W/(m^2*str),
    VisSolflux=626 #(W/m^2) Solar constant, in band 400nm to 800nm, from spectralcalc.c
        #blackbody (approximation for sun)
    QE=0.65 #Quantum efficiency
    opttrans=0.9 #This value fixed. chosen based on low cost commercial telescopes ~0.7
    SNR=6 #Minimum signal to noise ratio permitting detection
    refl=.15 # reflectivity, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20070
    Nd=6 #electrons/pixel/sec, this is constant, based on GEODSS performance data
    Nr=12 #electrons/pixel, this is constant, based on GEODSS performance data
    avgwavelength=5.9*10**(-7) #(m) weighted average wavelength of bandpass using 5778k
        #400nm to 800nm (min to max nm)
    numObservers=3
    UBERList=[]#This is a list of lists of lists. The first list is the Intervals, the
    AllObservationIntervals=[]
    for g in xrange(0,Intervals):#Creates the list of time steps
        interval=0+g
        AllObservationIntervals.append(interval)
    UBERList.append(AllObservationIntervals)
    Counter=[0]*numTgt #this creates and initializes the Counter, which keeps track of
    UBERList.append(Counter)
```

```
    IDCounter=[0]*numTgt #this creates and Initialezes the ID COunter, which keeps trac
    UBERList.append(IDCounter)
    #PriorityList=[0]*numTgt #creates and initializes a list of priorities
    satCount=[0]*numTgt #this creates and initializes the list which holds how many tim

    PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4

    UBERList[1]=[3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3
    LPSolution=solution

    for i in xrange(0,Intervals): #travel down the intervals
        UBERList[1]=[x+1 for x in UBERList[1]] #this line increments the Counter for ea
        UBERList[2]=[x+1 for x in UBERList[2]] #this line increments the ID Counter for
        for s in xrange(0,numObservers):#travels down the sensors
            for j in xrange(0,numTgt): #travel down the objects
                if LPSolution[s][j][i] == 1:
                    UBERList[1][j] = 0 #resets the counter
                    satCount[j] = satCount[j] + 1

    #code for pareto
    penalty_neg=[5, 5, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 50, 3, 5, 0, 0, 0, 0, 0, 0, 0,
    penalty_pos=[0, 0, 0, 619, 0, 0, 47, 249, 1126, 0, 69, 0, 0, 0, 0, 0, 0, 2, 356, 0,
    resetTime=[0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
    leftside=0
    rightside=0
    for j in xrange(0,numTgt): #travel down the intervals
        temp=0
        for k in xrange(0,numObservers): #travel down the sensors
            for i in xrange(0,Intervals): #travel down the objects
                temp += w1*solution[k][j][i]*weights[j][0]
        leftside += temp-w1*penalty_pos[j]*weights[j][1]-w1*penalty_neg[j]*weights[j][2
        rightside += w2*AgeWeights[j]*1.0*resetTime[j]
    print 'leftside = ', leftside/w1
    print 'rightside = ', rightside/w2

    #%%
    stopT=time.time()
    runTime=(stopT-staT)
# For efficiency, if the null architecture shows up, skip the simulation and give it th
# The "time.sleep" command ensures that the null instance doesn't start the failed node
else:
    fitness=[(86400/60.0)]

count1A=PriorityList.count(1.1)
count1B=PriorityList.count(1.2)
count1C=PriorityList.count(1.3)
count1D=PriorityList.count(1.4)
count1E=PriorityList.count(1.5)
count2A=PriorityList.count(2.1)
count2B=PriorityList.count(2.2)
count2C=PriorityList.count(2.3)
count2D=PriorityList.count(2.4)
count2E=PriorityList.count(2.5)
count3A=PriorityList.count(3.1)
count3B=PriorityList.count(3.2)
```

```
count3C=PriorityList.count(3.3)
count3D=PriorityList.count(3.4)
count3E=PriorityList.count(3.5)
count4A=PriorityList.count(4.1)
count4B=PriorityList.count(4.2)
count4C=PriorityList.count(4.3)
count4D=PriorityList.count(4.4)
count4E=PriorityList.count(4.5)
count5A=PriorityList.count(5.1)
count5B=PriorityList.count(5.2)
count5C=PriorityList.count(5.3)
count5D=PriorityList.count(5.4)
count5E=PriorityList.count(5.5)

countSat = 0
for y in range(0, len(satCount)):
    countSat = countSat + satCount[y]


plat=platform.system()
os.chdir(workPath)
if plat=='Windows':
    #print 'Fitness: ', fitness
    print 'Runtime: ' +str(runTime)
    #print 'UberList: ', UBERList[1]
    #print 'SatCount: ', satCount
    print 'Total Count: ', countSat
    #print 'Priority List: ', PriorityList
    print '# each Cats: '
    print '1A: ', count1A
    print '1B: ', count1B
    print '1C: ', count1C
    print '1D: ', count1D
    print '1E: ', count1E
    print '2A: ', count2A
    print '2B: ', count2B
    print '2C: ', count2C
    print '2D: ', count2D
    print '2E: ', count2E
    print '3A: ', count3A
    print '3B: ', count3B
    print '3C: ', count3C
    print '3D: ', count3D
    print '3E: ', count3E
    print '4A: ', count4A
    print '4B: ', count4B
    print '4C: ', count4C
    print '4D: ', count4D
    print '4E: ', count4E
    print '5A: ', count5A
    print '5B: ', count5B
    print '5C: ', count5C
    print '5D: ', count5D
    print '5E: ', count5E
    print '1s: ', count1A+count1B+count1C+count1D+count1E
```

```
print '2s: ', count2A+count2B+count2C+count2D+count2E
print '3s: ', count3A+count3B+count3C+count3D+count3E
print '4s: ', count4A+count4B+count4C+count4D+count4E
print '5s: ', count5A+count5B+count5C+count5D+count5E

print 'Results: '
print 'Mean Age All: ', sum(UBERList[1])/len(UBERList[1])
print 'Max Age All: ', max(UBERList[1])
ones=[]
for i in xrange(0,len(UBERList[1])):
    if PriorityList[i]<2:
        ones.append(UBERList[1][i])
if ones==[]:
    print 'Mean Age of Cat 1: N/A'
    print 'Max Age of Cat 1: N/A'
else:
    print 'Mean Age of Cat 1: ', sum(ones)/len(ones)
    print 'Max Age of Cat 1: ', max(ones)
twos=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(UBERList[1][i])
if twos==[]:
    print 'Mean Age of Cat 2: N/A'
    print 'Max Age of Cat 2: N/A'
else:
    print 'Mean Age of Cat 2: ', sum(twos)/len(twos)
    print 'Max Age of Cat 2: ', max(twos)
threes=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(UBERList[1][i])
if threes==[]:
    print 'Mean Age of Cat 3: N/A'
    print 'Max Age of Cat 3: N/A'
else:
    print 'Mean Age of Cat 3: ', sum(threes)/len(threes)
    print 'Max Age of Cat 3: ', max(threes)
fours=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(UBERList[1][i])
if fours==[]:
    print 'Mean Age of Cat 4: N/A'
    print 'Max Age of Cat 4: N/A'
else:
    print 'Mean Age of Cat 4: ', sum(fours)/len(fours)
    print 'Max Age of Cat 4: ', max(fours)
fives=[]
for i in xrange(0,len(UBERList[1])):
    if (PriorityList[i]>5):
        fives.append(UBERList[1][i])
if fives==[]:
    print 'Mean Age of Cat 5: N/A'
    print 'Max Age of Cat 5: N/A'
```

```
else:
    print 'Mean Age of Cat 5: ', sum(fives)/len(fives)
    print 'Max Age of Cat 5: ', max(fives)
print 'Total Observed All: ', sum(satCount)
ones=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]<2:
        ones.append(satCount[i])
if ones==[]:
    print 'Total Observed of Cat 1: N/A'
else:
    print 'Total Observed of Cat 1: ', sum(ones)
twos=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<3) and (PriorityList[i]>2):
        twos.append(satCount[i])
if twos==[]:
    print 'Total Observed of Cat 2: N/A'
else:
    print 'Total Observed of Cat 2: ', sum(twos)
threes=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<4) and (PriorityList[i]>3):
        threes.append(satCount[i])
if threes==[]:
    print 'Total Observed of Cat 3: N/A'
else:
    print 'Total Observed of Cat 3: ', sum(threes)
fours=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]<5) and (PriorityList[i]>4):
        fours.append(satCount[i])
if fours==[]:
    print 'Total Observed of Cat 4: N/A'
else:
    print 'Total Observed of Cat 4: ', sum(fours)
fives=[]
for i in xrange(0,len(satCount)):
    if (PriorityList[i]>5):
        fives.append(satCount[i])
if fives==[]:
    print 'Total Observed of Cat 5: N/A'
else:
    print 'Total Observed of Cat 5: ', sum(fives)
threshold=[]
for i in xrange(0,len(satCount)):
    if PriorityList[i]==1.1:
        threshold.append(limit[0])
    elif PriorityList[i]==1.2:
        threshold.append(limit[1])
    elif PriorityList[i]==1.3:
        threshold.append(limit[2])
    elif PriorityList[i]==1.4:
        threshold.append(limit[3])
    elif PriorityList[i]==1.5:
```

```
        threshold.append(limit[4])
    elif PriorityList[i]==2.1:
        threshold.append(limit[5])
    elif PriorityList[i]==2.2:
        threshold.append(limit[6])
    elif PriorityList[i]==2.3:
        threshold.append(limit[7])
    elif PriorityList[i]==2.4:
        threshold.append(limit[8])
    elif PriorityList[i]==2.5:
        threshold.append(limit[9])
    elif PriorityList[i]==3.1:
        threshold.append(limit[10])
    elif PriorityList[i]==3.2:
        threshold.append(limit[11])
    elif PriorityList[i]==3.3:
        threshold.append(limit[12])
    elif PriorityList[i]==3.4:
        threshold.append(limit[13])
    elif PriorityList[i]==3.5:
        threshold.append(limit[14])
    elif PriorityList[i]==4.1:
        threshold.append(limit[15])
    elif PriorityList[i]==4.2:
        threshold.append(limit[16])
    elif PriorityList[i]==4.3:
        threshold.append(limit[17])
    elif PriorityList[i]==4.4:
        threshold.append(limit[18])
    elif PriorityList[i]==4.5:
        threshold.append(limit[19])
    elif PriorityList[i]==5.1:
        threshold.append(limit[20])
    elif PriorityList[i]==5.2:
        threshold.append(limit[21])
    elif PriorityList[i]==5.3:
        threshold.append(limit[22])
    elif PriorityList[i]==5.4:
        threshold.append(limit[23])
    else:
        threshold.append(limit[24])
madeThreshold=[]
for i in xrange(0,len(satCount)):
    if satCount[i]>=threshold[i]:
        madeThreshold.append(1)
    else:
        madeThreshold.append(0)
print 'Total Met Target Threshold All: ', sum(madeThreshold)
ones=[]
for i in xrange(0,len(madeThreshold)):
    if PriorityList[i]<2:
        ones.append(madeThreshold[i])
if ones==[]:
    print 'Total Met Target Threshold of Cat 1: N/A'
else:
```

```
            print 'Total Met Target Threshold of Cat 1: ', sum(ones)
        twos=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]<3) and (PriorityList[i]>2):
                twos.append(madeThreshold[i])
        if twos==[]:
            print 'Total Met Target Threshold of Cat 2: N/A'
        else:
            print 'Total Met Target Threshold of Cat 2: ', sum(twos)
        threes=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]<4) and (PriorityList[i]>3):
                threes.append(madeThreshold[i])
        if threes==[]:
            print 'Total Met Target Threshold of Cat 3: N/A'
        else:
            print 'Total Met Target Threshold of Cat 3: ', sum(threes)
        fours=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]<5) and (PriorityList[i]>4):
                fours.append(madeThreshold[i])
        if fours==[]:
            print 'Total Met Target Threshold of Cat 4: N/A'
        else:
            print 'Total Met Target Threshold of Cat 4: ', sum(fours)
        fives=[]
        for i in xrange(0,len(madeThreshold)):
            if (PriorityList[i]>5):
                fives.append(madeThreshold[i])
        if fives==[]:
            print 'Total Met Target Threshold of Cat 5: N/A'
        else:
            print 'Total Met Target Threshold of Cat 5: ', sum(fives)
    else:
        workPath = os.environ['LOC']
        workSpace = os.environ['WORKDIR']
        repPath = os.path.join(workSpace,'Reports_Jan')
        scorePath = os.path.join(workSpace,'Jan',trial_num,'scores')
        os.chdir(scorePath)
        fin=open('Score'+str(numInst)+'.txt','w',os.O_NONBLOCK)
        #below are the penalty parameters and the gradient of the second tier of the penaly
        valMaxSize=75
        valMaxLat=90
        valMaxCost=30
        gradSize=valMaxSize*1.1
        gradLat=valMaxLat*1.1
        gradCost=valMaxCost*1.1
        #Here is where the penalty comes in, after the score is computed then it is accesse
        if fitness[0]>gradSize or fitness[1]>gradLat or fitness[2]>gradCost:
            fitness[0]=100000
            fitness[1]=100000
            fitness[2]=100000
```

1.2.8 Binary Integer Program Model Three Sensor Python Code

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""
import time
import platform

#Given Info
PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4.5,2

numTgt=50
UBERList = [[],[],[]]
UBERList.append(givenAvailability)

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]
m=2880 #Intervals

staT=time.time()
plat=platform.system()
    #%%
solution=[]
# Import PuLP modeler functions
from pulp import *
given = UBERList[3]#[[1,1,1,1,1],[0,1,1,0,0],[0,0,1,1,0]]
lpCols = len(given[0][0]) #intervals
lpRows = len(given[0]) #num tgt
lpSens = len(given) #sensors
Rows = []
Cols = []
Sens = []
Track=[]
for i in xrange(1,lpCols+1):
    Cols.append(i)
for i in xrange(1,lpRows+1):
    Rows.append(i)
for i in xrange(1,lpSens+1):
    Sens.append(i)
for i in xrange(0,lpRows):
    if PriorityList[i]==1.1:
        temp=limit[0]
    elif PriorityList[i]==1.2:
        temp=limit[1]
    elif PriorityList[i]==1.3:
        temp=limit[2]
    elif PriorityList[i]==1.4:
        temp=limit[3]
    elif PriorityList[i]==1.5:
        temp=limit[4]
```

1

```
    elif PriorityList[i]==2.1:
        temp=limit[5]
    elif PriorityList[i]==2.2:
        temp=limit[6]
    elif PriorityList[i]==2.3:
        temp=limit[7]
    elif PriorityList[i]==2.4:
        temp=limit[8]
    elif PriorityList[i]==2.5:
        temp=limit[9]
    elif PriorityList[i]==3.1:
        temp=limit[10]
    elif PriorityList[i]==3.2:
        temp=limit[11]
    elif PriorityList[i]==3.3:
        temp=limit[12]
    elif PriorityList[i]==3.4:
        temp=limit[13]
    elif PriorityList[i]==3.5:
        temp=limit[14]
    elif PriorityList[i]==4.1:
        temp=limit[15]
    elif PriorityList[i]==4.2:
        temp=limit[16]
    elif PriorityList[i]==4.3:
        temp=limit[17]
    elif PriorityList[i]==4.4:
        temp=limit[18]
    elif PriorityList[i]==4.5:
        temp=limit[19]
    elif PriorityList[i]==5.1:
        temp=limit[20]
    elif PriorityList[i]==5.2:
        temp=limit[21]
    elif PriorityList[i]==5.3:
        temp=limit[22]
    elif PriorityList[i]==5.4:
        temp=limit[23]
    else:
        temp=limit[24]
    Track.append(temp)

# The prob variable is created to contain the problem data
prob = LpProblem("IP Problem",LpMaximize)

# The problem variables are created
choices = LpVariable.dicts("Choice",(Sens,Rows,Cols),0,1,LpInteger)
penalty_pos = LpVariable.dicts("Over",(Rows),0,m,LpInteger)
penalty_neg = LpVariable.dicts("Under",(Rows),0,m,LpInteger)

weights=[]
for i in xrange(0,lpRows):
    if PriorityList[i]<2:
        weights.append([1,0.6,0.5])
    elif PriorityList[i]<3:
```

```
        weights.append([1,0.7,0.4])
    elif PriorityList[i]<4:
        weights.append([1,0.8,0.3])
    elif PriorityList[i]<5:
        weights.append([1,0.9,0.2])
    else:
        weights.append([1,0.95,0.1])

# The objective function is added
prob += lpSum((choices[s][r][c]*weights[r-1][0] for c in Cols for s in Sens)-penalty_pc

for s in Sens:
    for c in Cols:
        prob += lpSum(choices[s][r][c] for r in Rows) <= 3,""

for r in Rows:
    prob += lpSum((choices[s][r][c] for c in Cols for s in Sens)-penalty_pos[r]+penalty

# The starting numbers are entered as constraints
for z in xrange(0,lpSens):
    for x in xrange(0,lpRows):
        for y in xrange(0,lpCols):
            if given[z][x][y] == 0:
                prob += choices[z+1][x+1][y+1] == 0,""

# The problem data is written to an .lp file
prob.writeLP("model_lp_out.lp")

# A file called model_lp_out.txt is created/overwritten for writing to
model_lp_out = open('model_lp_out.txt','w')
solCount = 0
#while True:
prob.solve()

# The status of the solution is printed to the screen
print "\n","Status:", LpStatus[prob.status]
# The solution is printed if it was deemed "optimal" i.e met the constraints
if LpStatus[prob.status] == "Optimal":
    solCount += 1
    # The solution is written to the model_lp_out.txt file
    model_lp_out.write("[")
    for s in Sens:
        model_lp_out.write("[")
        pseudoptwoSol=[]
        for r in Rows:
            model_lp_out.write("[")
            pseudoSol=[]
            for c in Cols:
                    pseudoSol.append(int(value(choices[s][r][c])))
                    vee = str(int(value(choices[s][r][c])))
                    model_lp_out.write(vee)
                    if (c != lpCols):
                        model_lp_out.write(", ")
                    if c == lpCols:
                        model_lp_out.write("]")
```

```
                    if (c == lpCols) and (r != lpRows):
                        model_lp_out.write(", ")
                pseudoptwoSol.append(pseudoSol)
            model_lp_out.write("]")
            solution.append(pseudoptwoSol)
        model_lp_out.write("]")
        model_lp_out.write("\n+-------+-------+-------+\n\n")

        # Each of the variables is printed with it's resolved optimum value
#        for v in prob.variables():
#            print v.name, "=", v.varValue
        print "Objective Value = ", value(prob.objective)
        print "m Value = ", len(UBERList[3])
model_lp_out.close()
# The location of the solutions is give to the user
print "Solutions Written to model_lp_out.txt\n"
#print solution
stopT=time.time()
runTime=(stopT-staT)
print 'Runtime: ' +str(runTime)
```

1.2.9 Multi-Objective Binary Integer Program Model

```
"""
Created on Fri Aug 19 10:57:46 2016

@author: Wachtel
Modified by: KDararutana 2 Feb 2019

This script will complete the task of creating a schedule.  It has basic
priority logic built it focus a single sensor per time step on a specified
target.
"""
import time
import platform

#Given Info
PriorityList=[5.3,4.3,5.4,4.1,5.4,4.1,1.3,2.1,2.4,5.5,4.1,5.5,5.4,4.1,5.4,2.3,5.3,4.5,2

numTgt=50
UBERList = [[]]
UBERList.append([3749,4948,1696,2595,1541,5744,5510,3042,3394,188,4001,4822,1658,2176,3
UBERList.append([])
UBERList.append(givenAvailability)

#Snowy Tables
limit=[50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1,50,10,5,3,1]
m=2880 #Intervals

w1=0.5
w2=0.5

staT=time.time()
plat=platform.system()
    #%%
solution=[]
# Import PuLP modeler functions
from pulp import *
given = UBERList[3]#[[1,1,1,1,1],[0,1,1,0,0],[0,0,1,1,0]]
lpCols = len(given[0][0]) #intervals
lpRows = len(given[0]) #num tgt
lpSens = len(given) #sensors
Rows = []
Cols = []
Sens = []
Track=[]
for i in xrange(1,lpCols+1):
    Cols.append(i)
for i in xrange(1,lpRows+1):
    Rows.append(i)
for i in xrange(1,lpSens+1):
    Sens.append(i)
for i in xrange(0,lpRows):
    if PriorityList[i]==1.1:
        temp=limit[0]
    elif PriorityList[i]==1.2:
        temp=limit[1]
    elif PriorityList[i]==1.3:
```

1

```
        temp=limit[2]
    elif PriorityList[i]==1.4:
        temp=limit[3]
    elif PriorityList[i]==1.5:
        temp=limit[4]
    elif PriorityList[i]==2.1:
        temp=limit[5]
    elif PriorityList[i]==2.2:
        temp=limit[6]
    elif PriorityList[i]==2.3:
        temp=limit[7]
    elif PriorityList[i]==2.4:
        temp=limit[8]
    elif PriorityList[i]==2.5:
        temp=limit[9]
    elif PriorityList[i]==3.1:
        temp=limit[10]
    elif PriorityList[i]==3.2:
        temp=limit[11]
    elif PriorityList[i]==3.3:
        temp=limit[12]
    elif PriorityList[i]==3.4:
        temp=limit[13]
    elif PriorityList[i]==3.5:
        temp=limit[14]
    elif PriorityList[i]==4.1:
        temp=limit[15]
    elif PriorityList[i]==4.2:
        temp=limit[16]
    elif PriorityList[i]==4.3:
        temp=limit[17]
    elif PriorityList[i]==4.4:
        temp=limit[18]
    elif PriorityList[i]==4.5:
        temp=limit[19]
    elif PriorityList[i]==5.1:
        temp=limit[20]
    elif PriorityList[i]==5.2:
        temp=limit[21]
    elif PriorityList[i]==5.3:
        temp=limit[22]
    elif PriorityList[i]==5.4:
        temp=limit[23]
    else:
        temp=limit[24]
    Track.append(temp)

# The prob variable is created to contain the problem data
prob = LpProblem("IP Problem",LpMaximize)

# The problem variables are created
choices = LpVariable.dicts("Choice",(Sens,Rows,Cols),0,1,LpInteger)
penalty_pos = LpVariable.dicts("Over",(Rows),0,m,LpInteger)
penalty_neg = LpVariable.dicts("Under",(Rows),0,m,LpInteger)
resetTime = LpVariable.dicts("Choice",(Rows),0,1,LpInteger)
```

```
weights=[]
for i in xrange(0,lpRows):
    if PriorityList[i]<2:
        weights.append([1,0.6,0.5])
    elif PriorityList[i]<3:
        weights.append([1,0.7,0.4])
    elif PriorityList[i]<4:
        weights.append([1,0.8,0.3])
    elif PriorityList[i]<5:
        weights.append([1,0.9,0.2])
    else:
        weights.append([1,0.95,0.1])

AgeWeights=[]
maxAge=max(UBERList[1])*1.0#python does integer devision unless specifying decimals
for i in xrange(0,lpRows):
    if maxAge==0:
        AgeWeights.append(1)
    else:
        AgeWeights.append(380.8588524*UBERList[1][i]/maxAge)

# The objective function is added
prob += lpSum((w1*choices[s][r][c]*weights[r-1][0] for c in Cols for s in Sens)-w1*pena

for s in Sens:
    for c in Cols:
        prob += lpSum(choices[s][r][c] for r in Rows) <= 3,""

for r in Rows:
    prob += lpSum((choices[s][r][c] for c in Cols for s in Sens)-penalty_pos[r]+penalty

for s in Sens:
    for r in Rows:
        prob += lpSum(resetTime[r]) <= (choices[s][r][c] for c in Cols),""

# The starting numbers are entered as constraints
for z in xrange(0,lpSens):
    for x in xrange(0,lpRows):
        for y in xrange(0,lpCols):
            if given[z][x][y] == 0:
                prob += choices[z+1][x+1][y+1] == 0,""

# The problem data is written to an .lp file
prob.writeLP("model_lp_out.lp")

# A file called model_lp_out.txt is created/overwritten for writing to
model_lp_out = open('model_lp_out.txt','w')
solCount = 0
#while True:
prob.solve()

# The status of the solution is printed to the screen
print "\n","Status:", LpStatus[prob.status]
# The solution is printed if it was deemed "optimal" i.e met the constraints
```

```
if LpStatus[prob.status] == "Optimal":
    solCount += 1
    # The solution is written to the model_lp_out.txt file
    model_lp_out.write("[")
    for s in Sens:
        model_lp_out.write("[")
        pseudoptwoSol=[]
        for r in Rows:
            model_lp_out.write("[")
            pseudoSol=[]
            for c in Cols:
                    pseudoSol.append(int(value(choices[s][r][c])))
                    vee = str(int(value(choices[s][r][c])))
                    model_lp_out.write(vee)
                    if (c != lpCols):
                        model_lp_out.write(", ")
                    if c == lpCols:
                        model_lp_out.write("]")
                    if (c == lpCols) and (r != lpRows):
                        model_lp_out.write(", ")
            pseudoptwoSol.append(pseudoSol)
        model_lp_out.write("]")
        solution.append(pseudoptwoSol)
    model_lp_out.write("]")
    model_lp_out.write("\n+-------+-------+-------+\n\n")
    model_lp_out.write("penalty_neg = [")
    for r in Rows:
        vee = str(int(value(penalty_neg[r])))
        model_lp_out.write(vee)
        if r!=lpRows:
            model_lp_out.write(", ")
        pseudoSol=[]
    model_lp_out.write("]\n\n")
    model_lp_out.write("penalty_pos = [")
    for r in Rows:
        vee = str(int(value(penalty_pos[r])))
        model_lp_out.write(vee)
        if r!=lpRows:
            model_lp_out.write(", ")
        pseudoSol=[]
    model_lp_out.write("]\n\n")
    model_lp_out.write("resetTime = [")
    for r in Rows:
        vee = str(int(value(resetTime[r])))
        model_lp_out.write(vee)
        if r!=lpRows:
            model_lp_out.write(", ")
        pseudoSol=[]
    model_lp_out.write("]\n\n")
    # Each of the variables is printed with it's resolved optimum value
#    for v in prob.variables():
#        print v.name, "=", v.varValue
    print "Objective Value = ", value(prob.objective)
    print "m Value = ", len(UBERList[3])
model_lp_out.close()
```

```
# The location of the solutions is give to the user
print "Solutions Written to model_lp_out.txt\n"
#print solution
stopT=time.time()
runTime=(stopT-staT)
print 'Runtime: ' +str(runTime)
```

Appendix B. Analysis Result Tables

The following contains tables of results from simulation runs.

2.1 3968 RSO Results

Table 11. 3698 RSO Runtimes (s)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
Summer Solstice	66.446	174.577	179.054	182.339
Vernal Equinox	35.118	49.248	47.941	92.013
Winter Solstice	26.191	160.887	141.906	165.938

Table 12. 3698 RSO Summer Solstice Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	8625	8640	8640	8640
CAT 1	7566	7566	7566	7566
CAT 2	8625	8640	8630	8625
CAT 3	8570	8618	8618	8570
CAT 4	8592	8640	8640	8632
CAT 5	8624	8640	8640	8640

Table 13. 3698 RSO Vernal Equinox Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	8640	8640	8640	8640
CAT 1	7566	7902	7902	7902
CAT 2	8631	8640	8640	8640
CAT 3	8570	8618	8618	8618
CAT 4	8640	8640	8640	8640
CAT 5	8640	8640	8640	8640

Table 14. 3698 RSO Winter Solstice Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	8640	8640	8640	8640
CAT 1	7707	7707	7707	7707
CAT 2	8631	8640	8631	8631
CAT 3	8570	8618	8570	8570
CAT 4	8632	8640	8640	8632
CAT 5	8640	8640	8640	8640

Table 15. 3698 RSO Summer Solstice Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	4557	5760	5407	5210
CAT 1	4518	3134	3195	3156
CAT 2	4544	5739	3740	3661
CAT 3	4789	5962	5962	5679
CAT 4	4620	5798	5798	5571
CAT 5	4508	5774	5774	5556

Table 16. 3698 RSO Vernal Equinox Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	4520	5771	5723	5722
CAT 1	4242	5462	5462	5462
CAT 2	4594	5689	5680	5680
CAT 3	4457	5962	5871	5870
CAT 4	4507	5798	5766	5766
CAT 5	4512	5774	5706	5705

Table 17. 3698 RSO Winter Solstice Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	4598	5595	5358	4886
CAT 1	4414	4217	4351	4232
CAT 2	4634	4752	4490	4396
CAT 3	4691	5962	5650	5159
CAT 4	4604	5798	5577	5023
CAT 5	4577	5774	5538	4973

Table 18. 3698 RSO Summer Solstice Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	924	924	924	924
CAT 1	6	924	123	35
CAT 2	160	0	801	776
CAT 3	42	0	0	8
CAT 4	237	0	0	37
CAT 5	479	0	0	68

Table 19. 3698 RSO Vernal Equinox Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	1218	1218	1218	1218
CAT 1	9	0	0	0
CAT 2	195	1218	323	210
CAT 3	66	0	76	89
CAT 4	325	0	156	173
CAT 5	623	0	663	746

Table 20. 3698 RSO Winter Solstice Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	1494	1494	1494	1494
CAT 1	8	588	104	26
CAT 2	248	906	1250	802
CAT 3	79	0	11	40
CAT 4	375	0	43	202
CAT 5	784	0	86	424

Table 21. 3698 RSO Summer Solstice Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	190	17	128	139
CAT 1	2	17	17	10
CAT 2	34	0	111	108
CAT 3	7	0	0	1
CAT 4	42	0	0	8
CAT 5	105	0	0	12

Table 22. 3698 RSO Vernal Equinox Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	231	10	42	40
CAT 1	2	0	0	0
CAT 2	42	10	10	6
CAT 3	12	0	1	2
CAT 4	48	0	5	5
CAT 5	127	0	26	27

Table 23. 3698 RSO Winter Solstice Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	222	98	163	205
CAT 1	2	9	9	2
CAT 2	33	89	125	91
CAT 3	12	0	2	4
CAT 4	54	0	10	32
CAT 5	121	0	17	76

Table 24. 3698 RSO Summer Solstice Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	4.79%	0.43%	3.23%	3.50%
CAT 1	5.88%	50.00%	50.00%	29.41%
CAT 2	4.84%	0.00%	15.81%	15.38%
CAT 3	3.78%	0.00%	0.00%	0.54%
CAT 4	4.03%	0.00%	0.00%	0.77%
CAT 5	5.24%	0.00%	0.00%	0.60%

Table 25. 3698 RSO Vernal Equinox Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	5.82%	0.25%	1.06%	1.01%
CAT 1	5.88%	0.00%	0.00%	0.00%
CAT 2	5.98%	1.42%	1.42%	0.85%
CAT 3	6.49%	0.00%	0.54%	1.08%
CAT 4	4.61%	0.00%	0.48%	0.48%
CAT 5	6.33%	0.00%	1.30%	1.35%

Table 26. 3698 RSO Winter Solstice Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing
All	5.59%	2.47%	4.11%	5.17%
CAT 1	5.88%	26.47%	26.47%	5.88%
CAT 2	4.70%	12.68%	17.81%	12.96%
CAT 3	6.49%	0.00%	1.08%	2.16%
CAT 4	5.18%	0.00%	0.96%	3.07%
CAT 5	6.03%	0.00%	0.85%	3.79%

2.2 190 RSO Results

Table 27. 190 RSO Runtimes (s)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
Summer Solstice	1.913	17.304	16.305	16.943	198.067	238.718
Vernal Equinox	6.048	18.959	19.061	21.485	196.013	255.718
Winter Solstice	5.854	26.727	29.496	22.831	334.067	219.092

Table 28. 190 RSO Summer Solstice Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	8625	8625	8625	8625	8625	8625
CAT 1	1483	1741	1477	1488	1785	1808
CAT 2	8512	8512	8512	8512	8512	8512
CAT 3	8428	8428	8428	8428	8428	8428
CAT 4	8625	8625	8625	8625	8625	8625
CAT 5	8118	8500	8404	8118	8118	8118

Table 29. 190 RSO Vernal Equinox Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	8625	8625	8625	8625	8625	8625
CAT 1	1337	1818	1334	1312	1635	1671
CAT 2	8512	8512	8512	8512	8512	8512
CAT 3	8428	8428	8428	8428	8428	8428
CAT 4	8625	8625	8625	8625	8625	8625
CAT 5	8500	8500	8500	8500	8500	8500

Table 30. 190 RSO Winter Solstice Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	8625	8626	8625	8625	8625	8625
CAT 1	1225	1224	1237	1222	1208	1208
CAT 2	8512	8626	8512	8512	8512	8512
CAT 3	8428	8428	8428	8428	8428	8428
CAT 4	8625	8625	8625	8625	8625	8625
CAT 5	8500	8500	8500	8500	8500	8500

Table 31. 190 RSO Summer Solstice Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	3973	5406	4043	3972	4037	4040
CAT 1	1479	1600	1476	1484	1666	1669
CAT 2	3218	4235	3220	3220	3307	3322
CAT 3	3363	5228	3365	3361	3418	3436
CAT 4	3943	5879	4044	3940	3998	4014
CAT 5	4354	5636	4441	4354	4414	4404

Table 32. 190 RSO Vernal Equinox Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	3687	5379	3741	3687	3773	3784
CAT 1	1335	1562	1331	1311	1504	1575
CAT 2	3279	4072	3281	3278	3405	3435
CAT 3	3401	5228	3399	3404	3528	3535
CAT 4	3696	5879	3840	3695	3798	3794
CAT 5	3896	5636	3930	3897	3954	3966

Table 33. 190 RSO Winter Solstice Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	3764	5591	3769	3746	3844	3846
CAT 1	1223	1208	1224	1219	1202	1202
CAT 2	3473	5432	3361	3362	3473	3466
CAT 3	3209	5228	3213	3206	3395	3367
CAT 4	3466	5879	3467	3466	3570	3582
CAT 5	4128	5636	4171	4128	4209	4213

Table 34. 190 RSO Summer Solstice Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	924	924	924	924	924	924
CAT 1	21	798	30	21	15	15
CAT 2	154	126	322	163	190	190
CAT 3	78	0	60	78	77	77
CAT 4	251	0	191	249	317	317
CAT 5	420	0	321	413	325	325

Table 35. 190 RSO Vernal Equinox Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	1218	1218	1218	1218	1218	1218
CAT 1	33	1017	39	35	15	15
CAT 2	233	201	474	241	269	269
CAT 3	89	0	66	87	77	77
CAT 4	293	0	213	289	280	280
CAT 5	570	0	426	566	577	577

Table 36. 190 RSO Winter Solstice Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	1494	1494	1494	1494	1494	1494
CAT 1	29	1494	37	29	192	192
CAT 2	271	0	474	279	254	254
CAT 3	125	0	102	125	77	77
CAT 4	416	0	347	413	333	333
CAT 5	653	0	534	648	638	638

Table 37. 190 RSO Summer Solstice Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	48	10	52	48	64	62
CAT 1	1	2	2	1	2	2
CAT 2	10	8	15	10	13	13
CAT 3	5	0	5	5	7	7
CAT 4	12	0	11	12	19	19
CAT 5	20	0	19	20	23	21

Table 38. 190 RSO Vernal Equinox Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	54	15	59	54	80	80
CAT 1	2	2	2	2	2	2
CAT 2	11	13	17	11	16	16
CAT 3	6	0	6	6	7	7
CAT 4	12	0	12	12	21	21
CAT 5	23	0	22	23	34	34

Table 39. 190 RSO Winter Solstice Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	55	2	59	55	81	81
CAT 1	1	2	2	1	1	1
CAT 2	10	0	16	10	15	15
CAT 3	5	0	5	5	7	7
CAT 4	17	0	16	17	23	23
CAT 5	22	0	20	22	35	35

Table 40. 190 RSO Summer Solstice Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	25.26%	5.26%	27.37%	25.26%	33.68%	32.63%
CAT 1	50.00%	100.00%	100.00%	50.00%	100.00%	100.00%
CAT 2	33.33%	26.67%	50.00%	33.33%	43.33%	43.33%
CAT 3	38.46%	0.00%	38.46%	38.46%	53.85%	53.85%
CAT 4	25.00%	0.00%	22.92%	25.00%	39.58%	39.58%
CAT 5	20.62%	0.00%	19.59%	20.62%	23.71%	21.65%

Table 41. 190 RSO Vernal Equinox Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	28.42%	7.89%	31.05%	28.42%	42.11%	42.11%
CAT 1	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
CAT 2	36.67%	43.33%	56.67%	36.67%	53.33%	53.33%
CAT 3	46.15%	0.00%	46.15%	46.15%	53.85%	53.85%
CAT 4	25.00%	0.00%	25.00%	25.00%	43.75%	43.75%
CAT 5	23.71%	0.00%	22.68%	23.71%	35.05%	35.05%

Table 42. 190 RSO Winter Solstice Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	28.95%	1.05%	31.05%	28.95%	42.63%	42.63%
CAT 1	50.00%	100.00%	100.00%	50.00%	50.00%	50.00%
CAT 2	33.33%	0.00%	53.33%	33.33%	50.00%	50.00%
CAT 3	38.46%	0.00%	38.46%	38.46%	53.85%	53.85%
CAT 4	35.42%	0.00%	33.33%	35.42%	47.92%	47.92%
CAT 5	22.68%	0.00%	20.62%	22.68%	36.08%	36.08%

2.3 50 RSO Results

Table 43. 50 RSO Multi Sensor Runtimes (s)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
Summer Solstice	2.272	15.601	13.683	11.671	222.718	238.718
Vernal Equinox	5.072	22.173	22.642	16.575	225.718	234.718
Winter Solstice	5.123	20.533	15.890	15.025	235.718	238.718

Table 44. 50 RSO Multi Sensor Summer Solstice Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	7828	8624	7828	7828	7828	7828
CAT 1	1179	7739	1180	1179	1179	1179
CAT 2	3133	8095	3133	3133	3133	3133
CAT 3	3349	6954	3349	3349	3349	3349
CAT 4	7828	8624	7828	7828	7828	7828
CAT 5	6629	8500	6629	6629	6629	6629

Table 45. 50 RSO Multi Sensor Vernal Equinox Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	6629	8624	6629	6629	6629	6629
CAT 1	3	1648	3	3	0	0
CAT 2	3133	8095	3133	3133	3133	3133
CAT 3	3349	6954	3349	3349	3349	3349
CAT 4	6025	8624	6025	6025	6025	6025
CAT 5	6629	8500	6629	6629	6629	6629

Table 46. 50 RSO Multi Sensor Winter Solstice Max Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	8624	8624	8624	8624	8624	8624
CAT 1	1087	8390	1085	1087	1085	1085
CAT 2	5412	8095	5412	5412	5412	5412
CAT 3	6954	6954	6954	6954	6954	6954
CAT 4	8624	8624	8624	8624	8624	8624
CAT 5	6629	8500	6629	6629	6629	6629

Table 47. 50 RSO Multi Sensor Summer Solstice Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	1899	4211	1899	1899	2085	2062
CAT 1	659	3224	659	659	659	659
CAT 2	857	2676	857	857	856	856
CAT 3	1577	3304	1577	1578	1598	1598
CAT 4	2050	4387	2050	2050	2118	2127
CAT 5	2232	4724	2232	2231	2614	2551

Table 48. 50 RSO Multi Sensor Vernal Equinox Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	1521	3835	1521	1521	1691	1701
CAT 1	2	1043	2	2	0	0
CAT 2	1004	2353	1004	1004	1002	1002
CAT 3	1570	3685	1570	1570	1568	1568
CAT 4	1580	3991	1580	1580	1595	1694
CAT 5	1755	4413	1755	1755	2148	2087

Table 49. 50 RSO Multi Sensor Winter Solstice Mean Age (30 sec intervals)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	2048	5287	2048	2048	2231	2135
CAT 1	973	4609	972	973	971	971
CAT 2	1940	4880	1940	1940	1939	1939
CAT 3	2542	4716	2542	2542	2541	2541
CAT 4	1887	5254	1888	1887	2097	2031
CAT 5	2248	5577	2248	2248	2506	2332

Table 50. 50 RSO Multi Sensor Summer Solstice Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	7272	7272	7272	7272	7272	7272
CAT 1	46	495	58	46	165	165
CAT 2	731	4179	827	733	3219	3218
CAT 3	301	189	286	301	945	942
CAT 4	2880	1201	2876	2879	2838	2842
CAT 5	3314	1208	3225	3313	105	105

Table 51. 50 RSO Multi Sensor Vernal Equinox Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	7776	7776	7776	7776	7776	7776
CAT 1	164	1908	176	164	677	677
CAT 2	998	5406	1095	1000	3638	3638
CAT 3	265	10	257	265	990	990
CAT 4	2933	294	2935	2932	2284	2284
CAT 5	3416	158	3313	3415	187	187

Table 52. 50 RSO Multi Sensor Winter Solstice Total Observations

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	8232	8232	8232	8232	8232	8232
CAT 1	678	4785	684	678	1595	1595
CAT 2	601	3447	689	602	1299	1299
CAT 3	672	0	662	671	1767	1765
CAT 4	3180	0	3144	3180	3093	3092
CAT 5	3101	0	3053	3101	478	481

Table 53. 50 RSO Multi Sensor Summer Solstice Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	35	29	35	35	36	36
CAT 1	2	2	2	2	2	2
CAT 2	5	5	5	5	5	5
CAT 3	2	2	2	2	2	2
CAT 4	12	10	12	12	13	13
CAT 5	14	10	14	14	14	14

Table 54. 50 RSO Multi Sensor Vernal Equinox Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	38	33	39	38	39	39
CAT 1	2	2	2	2	2	2
CAT 2	4	5	5	4	5	5
CAT 3	2	1	2	2	2	2
CAT 4	14	12	14	14	14	14
CAT 5	16	13	16	16	16	16

Table 55. 50 RSO Multi Sensor Winter Solstice Total RSO's Meeting Observation Threshold (RSO)

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	29	4	31	29	33	33
CAT 1	1	2	2	1	2	2
CAT 2	3	2	4	3	3	3
CAT 3	2	0	2	2	2	2
CAT 4	11	0	11	11	12	12
CAT 5	12	0	12	12	14	14

Table 56. 50 RSO Multi Sensor Summer Solstice Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	70.00%	58.00%	70.00%	70.00%	72.00%	72.00%
CAT 1	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
CAT 2	83.33%	83.33%	83.33%	83.33%	83.33%	83.33%
CAT 3	66.67%	66.67%	66.67%	66.67%	66.67%	66.67%
CAT 4	66.67%	55.56%	66.67%	66.67%	72.22%	72.22%
CAT 5	66.67%	47.62%	66.67%	66.67%	66.67%	66.67%

Table 57. 50 RSO Multi Sensor Vernal Equinox Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	76.00%	66.00%	78.00%	76.00%	78.00%	78.00%
CAT 1	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
CAT 2	66.67%	83.33%	83.33%	66.67%	83.33%	83.33%
CAT 3	66.67%	33.33%	66.67%	66.67%	66.67%	66.67%
CAT 4	77.78%	66.67%	77.78%	77.78%	77.78%	77.78%
CAT 5	76.19%	61.90%	76.19%	76.19%	76.19%	76.19%

Table 58. 50 RSO Multi Sensor Winter Solstice Total RSO's Meeting Observation Threshold %

	Base Greedy	SSN Scheduler	Relaxed SSN	Relaxed SSN w/ Spacing	Binary Integer Program	Multi-Objective
All	58.00%	8.00%	62.00%	58.00%	66.00%	66.00%
CAT 1	50.00%	100.00%	100.00%	50.00%	100.00%	100.00%
CAT 2	50.00%	33.33%	66.67%	50.00%	50.00%	50.00%
CAT 3	66.67%	0.00%	66.67%	66.67%	66.67%	66.67%
CAT 4	61.11%	0.00%	61.11%	61.11%	66.67%	66.67%
CAT 5	57.14%	0.00%	57.14%	57.14%	66.67%	66.67%

Bibliography

1. U. S. Joint Chiefs of Staff, "Joint Publication 3-14, Space Operations," tech. rep., U. S. Joint Chiefs of Staff, Washington DC, 2013.

2. U. S. Joint Chiefs of Staff, "Joint Publication 3-14, Space Operations," tech. rep., U. S. Joint Chiefs of Staff, Washington, DC, 2018.

3. N. L. Johnson, "AAS 10-011 Orbital Debris: The Growing Threat to Space Operations," in *33rd Annual Guidance and Control Conference*, (Houston TX), pp. 1–8, NASA Johnson Space Center, NASA Johnson Space Center, 2010.

4. E.-I. Croitoru and G. Oancea, "Satellite Tracking Using NORAD Two-Line Element Set Format," *Scientific Research and Education in the Air Force-AFASES*, vol. 1, pp. 423–431, 2016.

5. J. Branke, K. Deb, and K. Miettinen, *Multiobjective Optimization: Interactive and evolutionary approaches*. Berlin Heidelberg: Springer-Verlag, 2008.

6. USSTRATCOM, "Strategic Command Directive SD 505-1 Vol 2," tech. rep., USSTRATCOM, Washington DC, 2004.

7. D. Moomey, "A Call to Action: Aid Geostationary Space Situational Awareness with Commercial Telescopes," *Air and Space Power Journal*, no. Nov-Dec, pp. 12–30, 2015.

8. P. R. Author Mark Ackermann, P. C. Zimmer, J. McGraw, l. T. John McGraw, and l. D. David Cox, "A Systematic Examination of Ground-Based and Space-Based Approaches to Optical Detection and Tracking of Satellites," in *31st Space Symposium*, (Colorado Springs CO), 2015.

9. J. Stern and S. Wachtel, *Genetic Algorithm Optimization of Geosynchronous Earth Orbit Space Situational Awareness Systems via Parallel Evaluation of Executable Architectures*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 2017.

10. M. Felten, *Optimization of Geosynchronous Space Situational Awareness Architectures Using Parallel Computation*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 2018.

11. M. Bateman, *Optimization of Geosynchronous Earth Orbit and Ascent Vehicle Space Situational Awareness via Parallel Evaluation of Executable Architectures*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 2018.

12. G. H. Mccall and J. H. Darrah, "Senior Leader Perspective Space Situational Awareness Difficult, Expensive and Necessary," *Air & Space Power Journal*, no. Nov-Dec, pp. 6–16, 2014.

13. M. S. Balch, R. Martin, and S. Ferson, "Satellite Conjunction Analysis and the False Confidence Theorem," tech. rep., arXiv:1706.08565 [math.ST], 2017.

14. F. Sanson and C. Frueh, "Probability of Detection and Likelihood: Application To Space Object Observation," in *7th European Conference on Space Debris*, (Darmstadt Germany), ESA Space Debris Office, 2017.

15. D. Koblick, A. Goldsmith, M. Klug, P. Mangus, B. Flewelling, M. Jah, J. Shanks, R. Piña, J. Stauch, J. Baldwin, J. Campbell, L. Col, and T. Blake, "Ground Optical Signal Processing Architecture for Contributing Space-based SSA Sensor Data," in *Advanced Maui Optical and Space Surveillance Technologies (AMOS) Conference*, (Wailea HI), 2014.

16. W. Basraoui, *Analysis of Merit-Based Observation Scheduling for Geosynchronous Earth Orbit Space Situational Awareness*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 2018.

17. A. C. Snow, J. L. Worthy, A. den Boer, L. J. Alexander, M. J. Holzinger, and D. Spencer, "Optimization of CubeSat Constellations for Uncued Electrooptical Space Object Detection and Tracking," *Journal of Spacecraft and Rockets*, vol. 53, pp. 401–419, 5 2016.

18. E. M. Vallerie, *Investigation of Photometric Data Received from an Artificial Earth Satellite: Discovery Service for Air Force Institute of Technology*. Wright-Patterson AFB, OH: Air Force Institute of Technology, 1963.

19. H. Evans, J. Lange, and J. Schmitz, *The Phenomenology of Intelligence-Focused Remote Sensing*. Riverside Research, 2014.

20. K. Krisciunas and B. E. Schaefer, "A Model of the Brightness of Moonlight," *Publications of the Astronomical Society of the Pacific*, vol. 103, pp. 1033–1039, 1991.

21. G. H. Thomson and S. B. Howell, "Handbook of CCD Astronomy," *Imaging Science Journal*, vol. 52, no. 1, pp. 49–50, 2004.

22. W. Wiesel, *Spaceflight Dynamics*. New York NY: McGraw-Hill, 2nd edition ed., 1997.

23. J. Tennenbaum and B. Director, "How Gauss determined the orbit of Ceres," *Fidelio*, vol. 7, no. 2, pp. 4–88, 1998.

24. J. T. Horwood, A. B. Poore, and K. T. Alfriend, "Orbit Determination and Data Fusion in GEO," in *Advanced Maui Optical and Space Surveillance Technologies Conference*, (Wailea HI), 2011.

25. P. Bernstein, "Background Paper on Space Surveillance Network (SSN) Tasking," tech. rep., AFSPC/A5CI, 2013.

26. D. A. Vallado and J. D. Griesbach, "Simulating Space Surveillance Networks (AAS 11-580)," in *AAS/AIAA Astrodynamics Specialist Conference*, (Girdwood AK), pp. 2769–2788, AAS 11-580, Published for the American Astronautical Society by Univelt, 2011.

27. Air Command and Staff College, *AU-18 Space Primer*. Maxwell Air Force Base, Alabama: Air University Press, 2nd ed., 2009.

28. C. Albon, "Raytheon and LM to Further Define Proposals: Old Space Fence Shut Down; AFSPC Requests Studies For Stalled Follow-On," *Inside the Air Force*, vol. 24, no. 38, pp. 10–11, 2013.

29. Joint Force Space Component Command Public Affairs, "Combined Space Operations Center Established at Vandenberg AFB," 7 2018.

30. V. S. Tanaev, V. S. Gordon, and Y. M. Shafransky, *Scheduling Theory. Single-Stage Systems*. Dordrect The Netherlands: Kluwer Academic Publishers, 1994.

31. M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. New York NY: Springer Science+Business Media, LLC, fifth edition ed., 2016.

32. S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. E. Horwood, 1982.

33. T. D. Gooley, *Automating the Satellite Range Scheduling Process*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1993.

34. M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear Programming and Network Flows*. Hoboken NJ: John Wiley & Sons, 4th edition ed., 2010.

35. J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge MA: MIT Press, 1992.

36. D. A. Parish, *A Genetic Algorithm Approach to Automating Satellite Range Scheduling*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1994.

37. A. Hinze and H. Fiedler, "Performance of Optimized Scheduled Follow-up Observations for Geosynchronous Space Objects Using Different Genetic Algorithms," in *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, (Wailea HI), 2017.

38. A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal*, vol. 3, no. 3, 1959.

39. E. Alpaydin, *Introduction to Machine Learning*. Cambridge MA: MIT Press, 3rd ed., 2014.

40. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge MA: MIT Press, 2nd ed., 2018.

41. D. J. Regan, "Modular Neural Network Tasking of Space Situational Awareness Systems," in *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, (Wailea HI), 2018.

42. R. Linares and R. Furfaro, "An Autonomous Sensor Tasking Approach for Large Scale Space Object Cataloging," in *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, (Wailea HI), 2017.

43. B. D. Little and C. Frueh, "SSA Sensor Tasking: Comparison of Machine Learning with Classical Optimization Methods," in *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, (Wailea, Maui, Hawaii), 2018.

44. C. A. Coello and N. Cruz Cortez, "An Approach to Solve Multiobjective Optimization Problems Based on an Artificial Immune System," in *First International Conference on Artificial Immune Systems*, 2003.

45. R. Hertwig and T. Pachur, "Heuristics, History of What Is a Heuristic?," in *International Encyclopedia of Social & Behavioral Sciences*, vol. 10, pp. 829–835, Berlin Germany: Elsevier LTD., 2nd edition ed., 2015.

46. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. Cambridge MA: MIT Press, 2009.

47. O. Grodzevich and O. Romanko, "Normalization and Other Topics in MultiObjective Optimization," in *Fields-MITACS Industrial Problems Workshop*, (Montreal, Canada), pp. 89–101, Fabien, 2006.

48. W. Basraoui, *Analysis of Merit-Based Observation Scheduling for Geosynchronous Earth Orbit Space Situational Awareness Air Force Institute of Technology*. MS thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 2018.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
21-03-2019	Master's Thesis	Sept 2017-Mar 2019

4. TITLE AND SUBTITLE

Comparison of Novel Heuristic and Integer Programming Schedulers for the USAF Space Surveillance Network

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dararutana, Kanit, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management
(AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-19-M-108

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Mark Storz HQ AFSPC/A36Z

150 Vandenberg St, Suite 1105, Peterson AFB,

10. SPONSOR/MONITOR'S ACRONYM(S)
AFSPC/A36Z

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Statement A. Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Space is a highly congested and contested domain begetting the importance of prioritizing the Space Situational Awareness (SSA) mission, especially that of scheduling and tasking the Space Surveillance Network (SSN). According to the 2004 USSTRATCOM Strategic Directive 505-1 (SD 505-1) the SSN uses centralized tasking, with decentralized scheduling. This research develops and compares novel scheduling models to a model reflecting the 2004 SD 505-1. Novel schedulers were developed to reduce time gaps between observations, prioritize high value space objects, and retain maximum observation quality. In both single and multi-sensor scenarios, these novel schedulers maintained the same, or higher, levels of observation threshold retention in high priority targets, while increasing observation threshold gains in lower categories. Simulations using the novel schedulers showed at least 3% improvement in meeting threshold requirements, 12% decrease in mean time between observations, and up to 9% decrease in maximum time between observations. Finally, these benefits were realized with a nominal increase in processing time for most novel schedulers. Results of this research can educate national policy makers on benefits of proposed upgrades to current and future SSA systems.

15. SUBJECT TERMS

Space Surveillance Network, Optimization, Integer Programming, Heuristics, Scheduler

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Lt Col Bruce A. Cox,AFIT/ENS
U	U	U	UU	366	**19b. TELEPHONE NUMBER** *(include area code)* (937)255-3636x4510 bruce.cox@afit.edu